WebSphere. software

IBM

IBM WebSphere eXtreme Scale V7: Solutions Architecture

- Product features
- Application scenarios
- Architectural overview

Ted Kirby
Jonathan Matthew
Gary Stone

Redpaper

ibm.com/redbooks

International Technical Support Organization

IBM WebSphere eXtreme Scale V7: Solutions Architecture

December 2009

REDP-4602-00

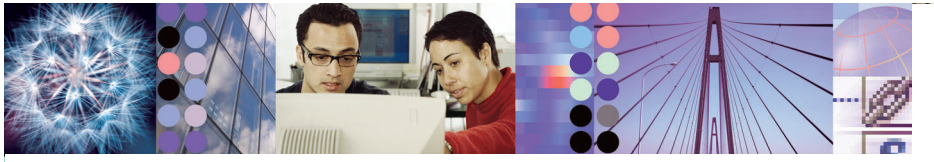
Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (December 2009)

This edition applies to IBM WebSphere eXtreme Scale V7.

© Copyright International Business Machines Corporation 2009. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contact an IBM Software Services Sales Specialist



Start SMALL, Start BIG, ... **JUST START**

architectural knowledge, skills, research and development . . .
that's IBM Software Services for WebSphere.

Our highly skilled consultants make it easy for you to design, build, test and deploy solutions, helping you build a smarter and more efficient business. **Our worldwide network of services specialists wants you to have it all!** Implementation, migration, architecture and design services: IBM Software Services has the right fit for you. We also deliver just-in-time, customized workshops and education tailored for your business needs. You have the knowledge, now reach out to the experts who can help you extend and realize the value.

For a WebSphere services solution that fits your needs, contact an IBM Software Services Sales Specialist:
ibm.com/developerworks/websphere/services/contacts.html

Contents

© Copyright IBM Corp. 2009. All rights reserved.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright IBM Corp. 2009. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

IBM®	Redbooks (logo) ®	WebSphere®
Rational®	solidDB®	
Redbooks®	Tivoli®	

The following terms are trademarks of other companies:

Hibernate, and the Shadowman logo are trademarks or registered trademarks of Red Hat, Inc. in the U.S. and other countries.

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

The IBM® WebSphere® eXtreme Scale product provides a powerful, elastic, high-performance, and scalable in-memory data grid. This paper will help IT architects understand how this data grid can be used to enhance application performance and scalability. It introduces the concepts behind eXtreme Scale and shows how it addresses the challenges of scalability and throughput found in today's business applications.

You will find information about entry points for integrating WebSphere eXtreme Scale into your environment, and a decision tree to help you select the features of eXtreme Scale that are especially suitable for improving performance in your environment.

This paper takes you through a number of application scenarios to illustrate the benefits that eXtreme Scale can provide. And finally, it provides an in-depth architectural discussion to help you understand how the product works and how it is integrated into an existing application environment.

This paper is a follow-on to *User's Guide to WebSphere eXtreme Scale*, SG24-7683, updating the architectural content for WebSphere eXtreme Scale V7. The technical portion of that book is still relevant and can be used as a guide to the implementation of eXtreme Scale.

The team who wrote this paper

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Ted Kirby is a Senior Software Engineer and a WebSphere Technical Evangelist for Extreme Transaction Processing at IBM in RTP, NC. He is an Apache Geronimo committer and was a WebSphere Application Server Community Edition developer. Previously, he has enhanced and maintained eCommerce Web sites and developed distributed operating systems, including the system used by the Deep Blue machine. Ted holds a BSE in Electrical Engineering and Computer Science from Princeton University and an MSE in Computer Science from UCLA.

© Copyright IBM Corp. 2009. All rights reserved.

Jonathan Matthew is a Staff Software Engineer with the Tivoli® Security development organization, working in Australia. He has over eight years of experience in IBM working on the Tivoli Access Manager family of products. He holds a degree in Software Engineering from the University of Queensland.

Gary Stone is a Software Engineer with IBM WebSphere Education, a team within IBM Software Services for WebSphere. Gary is responsible for course development and delivery for several WebSphere-based products including WebSphere eXtreme Scale and WebSphere Virtual Enterprise. Gary has also developed and taught courses for WebSphere Portal Application Development, Websphere Process Server and WebSphere Application Server. Prior to joining IBM, Gary was a member of the Education team at Transarc Corporation, which was acquired by IBM in 1999. Gary holds a BS in Computer Science from Clarion University of Pennsylvania.

Thanks to the following people for their contributions to this project:

Carla Sadtler
International Technical Support Organization, Raleigh Center

Billy Newport
IBM US

Debasish Banerjee
IBM US

Clara Liang
IBM US

Matthew Haynos
IBM US

Richard Szulewski
IBM US

Chris Johnson
IBM US

Jared Anderson
IBM US

Steve Branda
IBM US

Thanks to the authors of *User's Guide to WebSphere eXtreme Scale*, SG24-7683.

- Daniel Froehlich
- Nitin Gaur
- Jonathan Marshall
- John Pape
- Jennifer Zorza

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks® publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Introduction to WebSphere eXtreme Scale

This chapter describes some of the scalability challenges that exist in today's dynamic business and IT environments and how WebSphere eXtreme Scale addresses these challenges. An introduction of WebSphere eXtreme Scale and the key features of the product is also provided.

© Copyright IBM Corp. 2009. All rights reserved.

1.1 Scalability and throughput challenge

To understand the scalability challenge addressed by WebSphere eXtreme Scale, let us first define and understand scalability in a more traditional sense. Scalability is the ability of a system to handle increasing load in a graceful manner. This implies that a system can be readily extended. For example, a system has linear scaling capabilities so that doubling the CPU capacity also doubles the maximum throughput that the system can handle. In general, there are two ways an IT system can be scaled:

- Horizontally, by adding additional hosts to a tier. This is also called *scale out*.
- Vertically, by enlarging the capabilities of a single system. For example, adding CPUs. This is also called *scale up*.

Consider a classical three tier application such as the one shown in Figure 1-1. The application server tier is both scaled out by having three hosts and scaled up by having three application servers on each host. The database tier is scaled up by using a single powerful machine with many CPUs. The database tier is scaled out by having a shadow database using log shipping capability to support reports, analysis, and so forth.

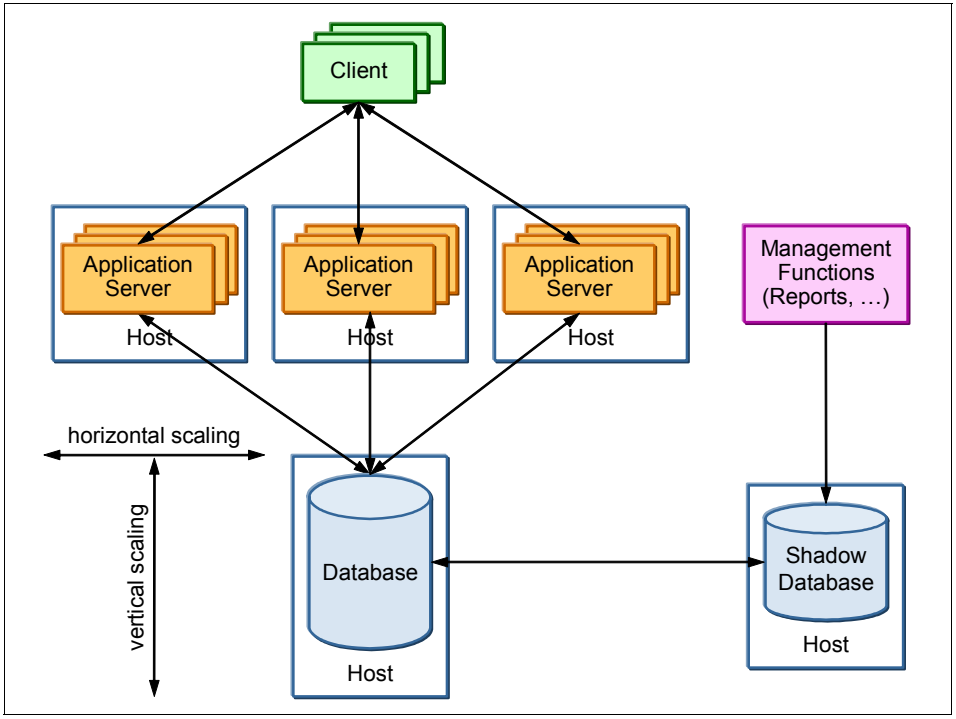


Figure 1-1 Scaling options in a traditional three tier application

Scaling is easy and effective as long as all of the resources involved can cope with the increased load. At some point a resource will reach its maximum throughput, thereby limiting the overall throughput of a system. This point is called the *saturation point* and the limiting resource is called a *bottleneck resource*.

Figure 1-2 shows the correlation between load and throughput that can typically be measured for an application.

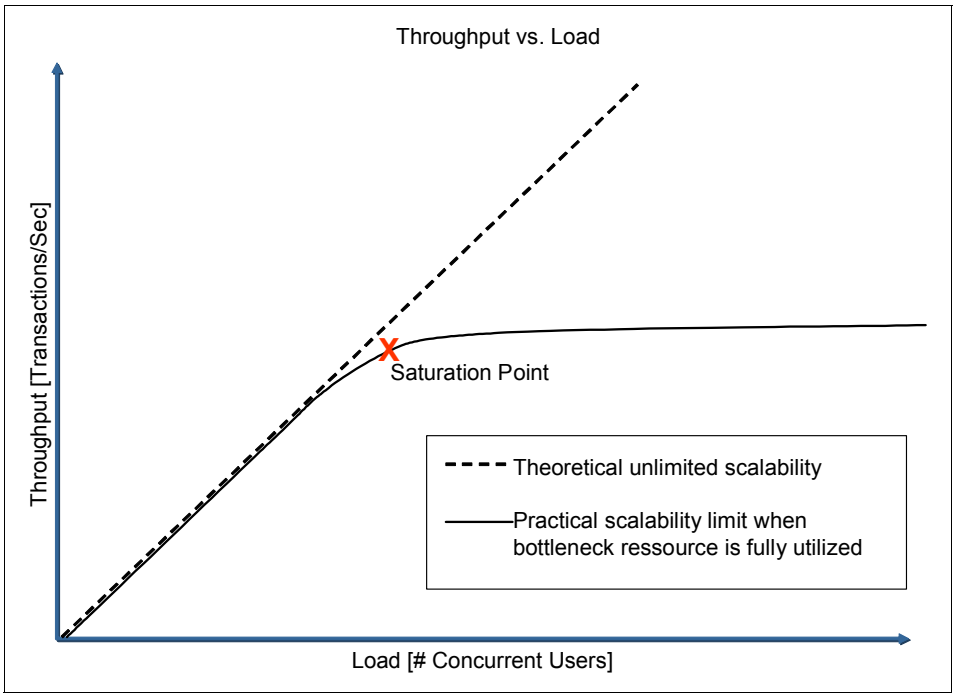


Figure 1-2 Correlation between throughput and load showing scalability limits

In a well-crafted application the database is usually the bottleneck resource. This is due to the fact that application servers can be effectively scaled horizontally to provide additional memory and processing cycles to service requests. Since the primary resource is the database, some of this additional load is passed on to the back-end database.

When the load on the database increases, the usual response is to scale up the database server also. At some point, either due to practical, financial, or physical limits, the database server is unable to continue to scale up. The progressive approach adopted is to scale out by adding additional database servers and

using a high speed connection between them to provide a cluster of database servers. This approach, while viable, poses additional challenges in keeping the database servers synchronized.

It is important to ensure that the databases are kept synchronized for data integrity and crash recovery. For example, consider two concurrent transactions that modify the same row in the database. When these transactions are executed by different database servers, communication is required to ensure the atomic, consistent, isolated, and durable (ACID) attributes of database transaction are preserved. This communication can grow exponentially as the number of database servers increases, which ultimately limits the scalability of the database. In fact, while application server clusters with more than 100, or even 1000, hosts can be easily found, a database server cluster with more then four members is hard to find.

The scalability challenge then, is to provide scalable access to large amounts of data. In almost all application scenarios, scalability is treated as a competitive advantage. It directly impacts the business applications and the business unit that owns the applications. This is because applications that are scalable can easily accommodate growth and aid the business functions in analysis and business development. You want to be able to scale your product with predictable costs, and without requiring a redeployment of an application or topology when the business grows.

1.1.1 Using caching solutions to improve performance and scalability

A typical approach to resolving performance and therefore scalability problems is to implement some form of caching of the data. Simply defined, the cache is a copy of frequently accessed data that is held in memory to reduce the access time to the data. This has the effect of placing the data closer to the application, which improves performance and throughput. It also reduces the number of requests to the database and thus reduces that resource's potential as a bottleneck in the application.

A cache, then, can simply extend the storage capability. It can be considered to act as a shock absorber to the database. As shown in Figure 1-3 on page 5, the cache sits between the application and the database to reduce the load on the database.

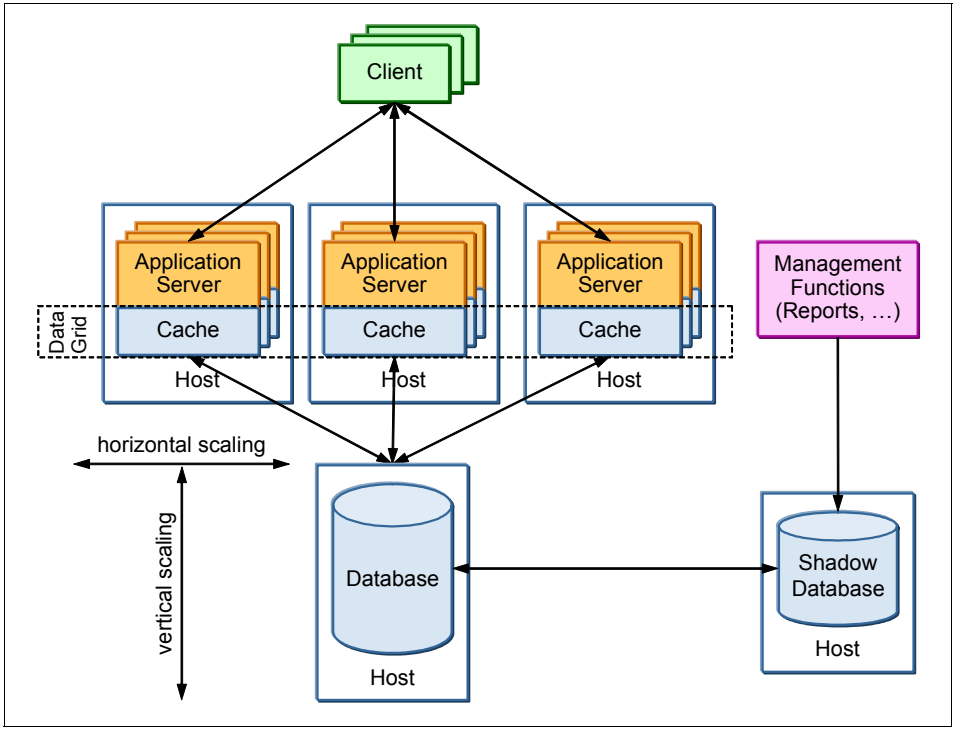


Figure 1-3 Introduce caching as response to the scalability challenge

While a cache can reduce the load on the database, the same data might be cached in several servers simultaneously. Things become complicated when one copy of the data is changed, because all cached copies need to be invalidated or updated. Taking the caching approach to the extreme leads to the idea of using data grids as a scalable solution.

Grid:	In general terms, a grid is a form of loosely-coupled and heterogeneous computers that act together to perform large tasks. To accomplish this task, a grid needs to be highly scalable. There are several different forms of grids, depending on the task at hand.
Data grid:	A data grid focuses on the provisioning and access of information in a grid style manner, that is, using a large amount of loosely-coupled cooperative caches to store data.

The data grid can be co-located with the applications as shown in Figure 1-3 on page 5. This provides the fastest access to the data. However, as the data grid grows, it may not scale effectively because the application and grid share the same address space.

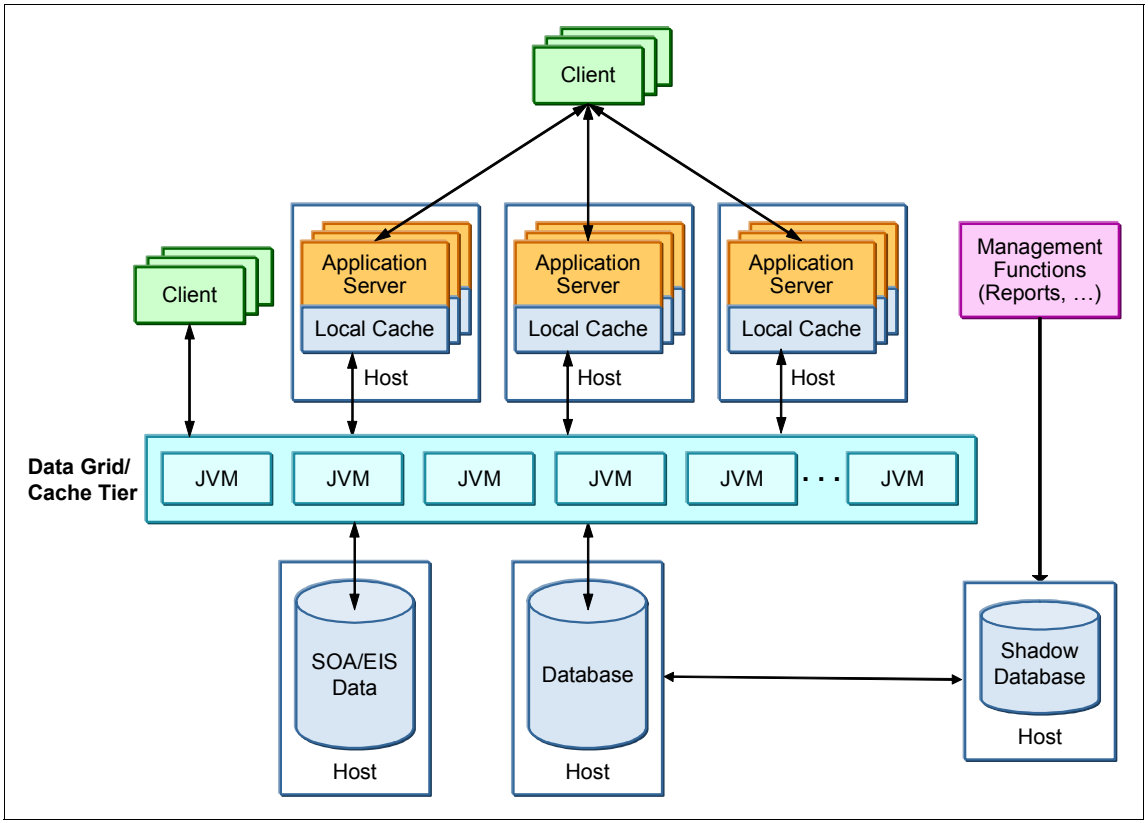


Figure 1-4 Separating the data grid to another tier

Another approach might be to perform local caching in the application server tier (near cache) and to have the data grid cache implementation be on a separate elastic tier where the servers are primarily only responsible for hosting the grid data. This architecture provides additional flexibility in designing the application topology as well as for the scalability and availability of the grid data.

As the use of the data grid technology increases it is possible that the cache or data grid becomes the primary data access point. In this case, the database is used by the grid as a long term persistent data store. If the back-end resource is unavailable, the applications are still able to complete transactions against the grid so there is no loss of service to the application. The grid can push those updates to the resource when it again becomes available. The back-end may

also be required in certain application scenarios where logging or auditing of transactions is required for compliance or by regulations. In some specialized use cases, such as transitory session state data, there may be no need to store the data from the grid into a back-end data store. Since the grid contains all of the information, data intensive computing tasks can then be moved into the grid and executed in parallel.

1.2 The WebSphere eXtreme Scale solution

WebSphere eXtreme Scale provides an extensible framework to simplify the caching of data used by an application. It provides solutions that range from a drop-in, out of the box cache, to a simple in-process cache, to the design of a highly scalable, fault tolerant data grid with nearly unlimited horizontal scaling capabilities. Solutions with WebSphere eXtreme Scale can also provide a quick time-to-value and a high return on investment (ROI) for the time and cost spent on implementation.

WebSphere eXtreme Scale enables infrastructure with the ability to deal with extreme levels of data processing and performance. When the data and resulting transactions experience incremental or exponential growth, the business performance does not suffer because the grid is easily and incrementally extended by adding additional capacity (Java™ virtual machines and hardware).

An eXtreme Scale grid is a viable solution to enterprise application scalability or performance issues. It is a solution that does not require specialized application server support or expensive hardware. There are also additional application architectures that can benefit from using the grid technology. The following sections introduces some of these example use cases.

1.2.1 Handling high performance extreme transaction processing

WebSphere eXtreme Scale is designed to handle and process very large in-memory datasets. Extreme transaction processing (XTP) can be defined as applications that have rigorous demands for scalability, performance, number of users supported, availability, and so forth.

An application could be considered an XTP application if:

- There is significant change expected in the number of users or transactions over the course of the lifetime of the application.
- If the current data volume is impractical to be cached in a single process.
- If there is a requirement to share data across multiple server instances.
- If there is a need to perform real-time analysis and processing of very large data sets (gigabytes) while maintaining consistently low response times.
- If there is a desire to collocate the application and the data together into one address space (JVM) to mitigate existing performance or scalability problems.
- If you have rigorous demands for 24x7 processing and applications that must be highly available using multiple data centers

1.2.2 Reducing back-end system load

In the case of implementing a cache or grid to provide the application with improved performance and scalability, the goal is to reduce the load (and potential bottleneck) on expensive back-end resources. An application can benefit from a grid solution, which eliminates a number of the redundant back-end calls, if the application exhibits some of the following characteristics:

- If performance testing and application profiling have shown that the back-end data source is the limiting factor to achieving the required performance or scalability goals.
- If the application is accessing the same data repeatedly.
- If the data is being shared and accessed by multiple applications simultaneously.
- If the back-end data resource operation is expensive performance-wise.
- If the current transaction rate is expected to double, triple, or grow by orders of magnitude while maintaining the current response times.
- If you have rigorous demands for 24x7 processing and applications that must be highly available using multiple data centers. WebSphere eXtreme Scale allows a common cache of data in geographically distributed environments with various levels of replication (synchronous and asynchronous) to provide timely and robust failover options. The support for failover and high-availability is built into the eXtreme Scale product and can be configured to provide the support required by each application.

1.2.3 Providing less expensive, more scalable cache solutions

Some application types can benefit from a type of drop-in cache solution for existing technologies that are used. In this case the application itself does not change or behave differently but the increased qualities of service that WebSphere eXtreme Scale provides are automatically made available to the application's cached data. Applications of this type are:

- Web applications that require automatic, fast, and reliable failover of HTTP session data in the event of a server failure. A Web banking or online shopping cart are examples of this type of application.
- Web applications that use the WebSphere Application Server Dynamic Cache API to cache data for subsequent requests.
- Client applications that use the ADO.NET REST data service interface and need to access data in a distributed grid.

1.2.4 Managing server state for application server farms

As distributed application environments evolve into a cloud and application server farm architectures, the need for very fast highly available storage is critical. WebSphere eXtreme scale can provide these features to a large stateless collection of application servers that process requests as required.

Due to the nature of the eXtreme Scale architecture, failover is transparent to the client because the conversational state is stored in the shared grid. The grid provides the data as requested to applications and provides the nominal highly available, scalable, and high capacity data storage qualities of service that are required. Applications that benefit from server farms have these types of characteristics:

- Applications where conversational state is not stored in the application server.
- Applications that require fast failover in the event of a JVM failure or shutdown.
- Applications with a need for an elastic amount of storage and a future need for significant storage growth is required.
- Applications where geographically distributed data centers potentially running in an active-active disaster recovery configuration are desired.

1.3 WebSphere eXtreme Scale product features

The WebSphere eXtreme Scale product provides the capabilities to solve the business problems described above.

The key features of WebSphere eXtreme Scale are:

- A highly available, scalable elastic grid
- Support for JSE, JEE, ADO.NET Data Services and REST capable client applications (REST is available in 7.0.0.0 cumulative fix 2).
- Transaction support
- Security
- Support for 3rd party monitoring solutions

1.3.1 Highly available, scalable elastic grids

As the product name suggests, WebSphere eXtreme Scale supports a dynamic grid infrastructure with support for substantial scale outs. It is designed to scale to thousands of server instances. This is possible by splitting large amounts of data into manageable chunks and distributing them across the grid.

Each of the server instances that is hosting grid data does so in a grid container. Experience has shown that large amounts of communication between containers in a grid can be a crucial limiting factor for scalability. This is why WebSphere eXtreme Scale grid containers have been designed to require little communication with each other, allowing large linear scale outs. Communication between grid containers is kept to a minimum, occurring only for availability management and data replication purposes.

WebSphere eXtreme Scale has been proven to run smoothly with more than 1500 JVMs with 2 Gb heap participating in a data grid managing almost 2 terabytes of data. The scale out was only limited by available hardware.

When the data in the grid must be highly available, the grid can be configured for replication so that in the event of a failure no loss of critical data occurs. The grid data is kept highly available by having multiple instances of the data stored on different servers, and even in different locations to ensure recovery in a disaster scenario. This capability is defined when the grid is created.

1.3.2 Support for JSE, JEE, ADO.NET Data Services, and REST applications

WebSphere eXtreme Scale does not require WebSphere Application Server as a runtime. It can use any native JSE (1.4+) or JEE application server environment. Not all features are available with JSE 1.4. JSE 1.5 or above adds additional capabilities with annotations and JPA support.

Some additional functionality for monitoring and maintaining the grid is available when eXtreme Scale is deployed in a WebSphere Application Server environment. It is also possible to access the data stored in the grid from an ADO.NET Data Services client using the Representational State Transfer (REST) API Data Services Framework. REST is available with 7.0.0.0 cumulative fix 2.

1.3.3 Transaction support

WebSphere eXtreme Scale has built-in transaction support for all changes made to the cached data. Changes are committed or rolled back in an atomic way. WebSphere eXtreme Scale can augment the database and acts as an intermediary between the application and database. It can also be the system of record for applications when no database or other back-end data store is used. Transaction processing ensures that multiple individual operations that work in tandem are treated as a single unit of work. If even one individual operation fails, the entire transaction fails.

As with other persistent store mechanisms, WebSphere eXtreme Scale uses transactions for the following reasons:

- To apply multiple changes as an atomic unit at commit time
- To ensure consistency of all cached data
- To isolate a thread from concurrent changes
- To act as the unit of replication to make the changes durable.
- To implement a life cycle for locks on changes
- To combine multiple changes to reduce number of remote invocations

1.3.4 Security

If the data that is stored in the cache is of a sensitive nature, security is an important point to consider. Similar to security for databases, fine-grained control over client access to data can be enforced.

WebSphere eXtreme Scale applications can enable security features and integrate with external security providers. A summary of the security features available includes:

- Authentication

  Authentication provides the ability to authenticate the identity of the client of the grid. This is done with credential information between the client and the grid server.

- Authorization

  Authorization provides an adequate level of access control to authenticated clients. The authorization includes operations such as reading, querying, and modifying the data in the grid.

- Transport security

  Transport security ensures secure communications between the remote clients and grid servers, and between the servers.

1.3.5 Support for monitoring solutions

Monitoring the availability and performance of enterprise data is critical for maintaining the integrity of the application infrastructure. WebSphere eXtreme Scale provides support for IBM Tivoli Monitoring (ITM) solution by providing an agent that integrates the two. Other third party monitoring products supported include CA Wily Introscope, and Hyperic HQ.

ITM agent and Hyperic HQ gather data from eXtreme Scale server side MBeans that are displayed on their consoles. CA Wily Introscope uses byte code instrumentation to display data. Sample PBD (Probe Builder Directive) files are provided to facilitate monitoring.

1.3.6 New features in V7

WebSphere eXtreme Scale version 7.0 has added many new features to improve the performance and usability of the product. The features detailed in the sections that follow were introduced with the latest version of the product.

Byte array maps

In some instances an application may improve the performance by using the byte array maps option. Typically the value that is stored in the grid is a Java object. The byte array map allows the application to store the value in a serialized form instead of as a Java object. This can potentially increase the server-side performance by eliminating the need to serialize and de-serialize the object on

each reference during client to server communication and during replication. It also improves the amount of memory used by the grid when complex objects are used.

Request timeout

Clients can configure a retry time value when an operation to a container server fails. If a retry value of more than zero is set, the request will be retried until the timeout condition occurs or until a permanent failure is encountered.

Monitoring tools support

Support has been added to provide monitoring of grid applications by some IBM and third party monitoring tools. Currently, the supported products are CA Wily Introscope, IBM agent for Tivoli Monitoring (ITM), and Hyperic HQ. Monitoring the health of the grid container and the JVMs that host the catalog service is critical. This new support provides access to the health data through these monitoring products. Information is available about the catalog service grid status, container server response times, transaction commit times, and more.

Dynamic cache provider

This functionality has been discussed previously as a use-case for eXtreme Scale (see 3.4, “WebSphere dynamic cache service replacement” on page 31). It is now possible for applications that use the WebSphere dynamic cache APIs to seamlessly take advantage of the quality of service and performance improvements by using eXtreme Scale. Instead of each WebSphere server holding a copy of the entire dynamic cache the grid can hold a distributed copy. This greatly reduces the size of the runtime JVM space required as well as the server startup time.

Composite index support

Previous versions of WebSphere eXtreme Scale have allowed you to have multiple indexes on the data in the grid. The new composite index support allows an index to be created on multiple attributes. This can make searching with a complex query more efficient.

Template maps

Typically, once a grid is initialized new maps can not be created on it. The addition of the template map capability removes this limitation. New maps can now be added locally or to a distributed environment after the servers are running and activated.

REST data service support

The REST data service support functionality is available in 7.0.0.0 cumulative fix 2. The WebSphere eXtreme Scale REST data service is a Java HTTP service that implements Microsoft®'s ADO.NET Data Services. The REST data service allows any HTTP client to access a WebSphere eXtreme Scale 7.0 grid. For more information, see the following Web page:

http://www.ibm.com/developerworks/websphere/downloads/xs_rest_service.html

2

Approaches to implementation

After explaining the main features of WebSphere eXtreme Scale, we would like to discuss how it can be implemented into an existing IT infrastructure. We discuss the implementation by showing possible entry points and providing a decision tree based on typical problems an organization might face.

© Copyright IBM Corp. 2009. All rights reserved.

2.1 The advantages of adopting eXtreme Scale

This section describes the challenges that lead to the adoption of WebSphere eXtreme Scale.

2.1.1 Scalability of back-end systems

Database servers and other back-end systems such as mainframe services often form application scalability bottlenecks. These systems can only be scaled up by purchasing faster hardware, which is costly and requires that additional capacity is purchased well before it is needed to handle the application load. WebSphere eXtreme Scale offers caching strategies to reduce the load on back-end systems, enhancing the value provided by these important systems. Due to eXtreme Scale's elastic nature scale out and scale up can be done as application load increases.

Consider the case of a database, which can be scaled up only so far before it is a bottleneck. The problem is that the fundamental database paradigm does not scale. The shadow database in Figure 1-1 on page 2 hints at this lack of scalability. The key to linear scaling is a partitioned data model, which WebSphere eXtreme Scale provides. eXtreme Scale distributes the load over the available servers. Servers may be added dynamically. The critical use case for WebSphere eXtreme Scale is to provide linear scaling to go beyond the bounds of the inherently limited and unscalable database model.

2.1.2 Application availability

In addition to being difficult to scale, back-end systems are often a limiting factor for application availability. When the application depends on a single back-end server, any downtime or connectivity interruption necessarily affects the whole application.

With write-behind caching, WebSphere eXtreme Scale can de-couple the application from the database or other back-end system. All of the application data is available in the grid, and updates to the back-end are held within the grid until they can be sent to the back-end system. As a result, the back-end system can be taken down for maintenance without affecting the grid application.

2.1.3 Application cache scalability

Applications that make use of simple caches to store large amounts of regularly accessed data reach scalability limits when the cache expands to the size of the virtual and physical system memory. To expand any further, either the cache must overflow onto disk storage, or it must be shared between servers. Disk off-loading is the simpler solution, but it means that disk access speed becomes critical for application performance. If disk access speed is not adequate, this solution essentially trades a database bottleneck for a disk bottleneck.

WebSphere eXtreme Scale can be used as a shared in-memory cache that can use memory on many systems. A shared cache can grow to the total size of all available memory in the grid, without introducing disk access speed as a factor. Unifying the cache also reduces the overhead of regenerating cached information, since all application servers use the cached data generated by the first server that needs it.

2.1.4 Multiple data centers

High availability and disaster recovery are desirable properties for any application, and are often achieved by running the application across multiple data centers. In this scenario, application state data needs to be shared between the data centers so that if one data center fails, the application can continue to function. To reduce data transfer between data centers, and reduce latency for data access, the application data should be stored where it is most likely to be used, then replicated elsewhere.

Commonly, the application is limited to running in an active-passive configuration, in which only one data center is actually running the application, with the other holding a replica of the application state and back-end data sources, ready to be activated in case the first data center fails. This is a result of restrictions imposed by back-end systems that do not support active-active replication, or do so by introducing a conflict resolution process.

WebSphere eXtreme Scale supports active-active configurations when running in multiple data centers. When the grid containers are properly configured into zones, and the application uses local zone placement to ensure that data is stored within the zone in which it is being used, data is only transferred between data centers for replication purposes. A key advantage is that this replication transfer can happen asynchronously, so you get the advantages of failover to a remote data center without the cost of synchronous replication to remote data centers on every data update.

2.1.5 Server consolidation

Application memory usage is one of the limiting factors to consider when consolidating applications onto shared server hardware. If the total working set size for all applications exceeds the available memory, performance suffers greatly due to disk paging. Moving application data into an external shared cache trades network bandwidth and processing power for local memory usage. The access speeds to the shared cache over a fast local network are also much faster than would be incurred through a disk I/O cycle.

WebSphere eXtreme Scale can be used to create a large external cache to store application state, while also providing a smaller near cache for frequently accessed data, thus reducing the application's local memory footprint.

2.2 Comparing eXtreme Scale to in-memory databases

WebSphere eXtreme Scale bears some similarity to in-memory database (IMDB) products such as IBM solidDB®, in that it aims to reduce the cost of data access by storing it in system memory rather than on disk.

While a solidDB instance is limited to the physical memory of a single machine, WebSphere eXtreme Scale can support much larger in-memory databases by using a huge number of machines to build the grid. WebSphere eXtreme Scale offers elastic scalability by adding and removing grid containers as the needs of the application change.

A second significant difference is that WebSphere eXtreme Scale stores data in the form of Java objects, rather than rows in a relational data model.

2.3 Entry points for WebSphere eXtreme Scale

WebSphere eXtreme Scale provides two fundamental benefits: large, distributed cache, and a scalable, partitioned data model. Most applications can benefit from some form of caching. The benefits are reduced user response time, and reduced load on the backing disk, system, database, or service holding the data. Adopting or converting to a partitioned data model allows linear scaling way beyond the bounds of the traditional database paradigm.

Figure 2-1 shows three possible entry points for adopting WebSphere eXtreme Scale, which can be used in ways ranging from a simple in-process cache to an enterprise-wide distributed data grid. The diagram also implies a roadmap. An organization can start with one of the lower entry points and evolve from there to the higher levels of grid computing.

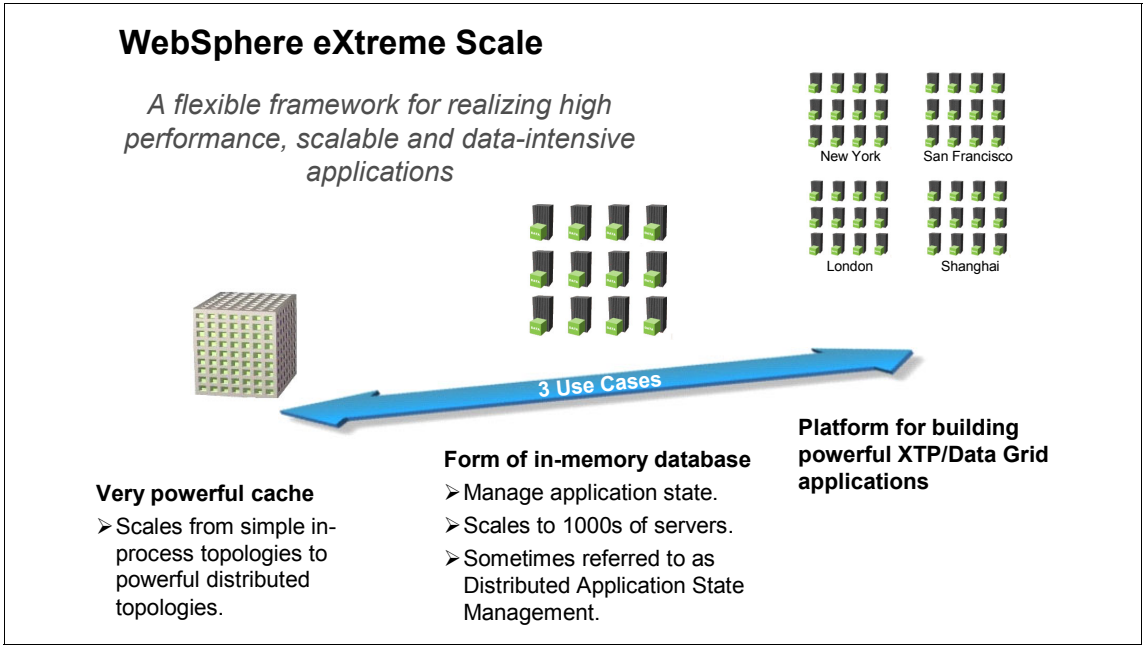


Figure 2-1 Entry points for adopting WebSphere eXtreme Scale

2.3.1 Entry point: side cache

The first entry point is the use of WebSphere eXtreme Scale as a sophisticated caching layer for an application. This proven and well supported IBM product provides an alternative to investing in the custom development of a home-grown solution, or to replace or augment an existing caching implementation to improve the scalability and security of the application, or to provide transactional data access where it is presently lacking. Only configuration changes to one or two XML configuration files are required for the application to exploit eXtreme Scale using this entry point. The use cases for this entry point are described in 3.3, “HTTP session state management” on page 28, 3.4, “WebSphere dynamic cache service replacement” on page 31, and 3.5, “Database caching” on page 32.

2.3.2 Entry point: eXtreme Scale as the system of record

The second entry point is the use of WebSphere eXtreme Scale as a data grid to store large amounts of data. A data grid can be incrementally extended, without interruption of service or data loss, to accommodate a great quantity of data, and just as easily scaled back in case the application requirements contract.

This entry point requires that you change your application code to use WebSphere eXtreme Scale. (The ADO.NET REST data service is a notable exception. You only need to change the data service URL to exploit eXtreme Scale in this case.) To implement this use, identify code areas where eXtreme Scale caching can help and insert code to use eXtreme Scale as a cache. The use case for this entry point is described in 3.6, “Caching for other back-end systems” on page 35.

One interesting special case where no application code changes are required is a service-oriented architecture (SOA) ESB mediation to cache results of SOA ESB calls for data.

2.3.3 Entry point: parallel processing with DataGrid APIs

The third entry point is the use of the extreme scalability of WebSphere eXtreme Scale to build an enterprise-wide complex data grid solution. It includes extensive use of grid-style computing, bringing the algorithms to the data rather than retrieving the required data from the back-end, as well as the option of unifying data stores between geographically distributed data centers.

In this entry point, you change your application to write to eXtreme Scale, not the database, so that eXtreme Scale is used as the system of record. This allows the exploitation of the write-behind cache. See 3.5, “Database caching” on page 32.

With data stored in the grid, the DataGrid APIs allow it to be processed in parallel. See 3.8, “Application processing in the grid” on page 38.

If you already have a data access layer that isolates and separates the data access calls from the rest of your business logic, it can be much easier to exploit eXtreme Scale as either a side-cache or an in-line cache. See 3.7, “Application data caching” on page 36.

2.4 Decision tree for adopting WebSphere eXtreme Scale

This section presents a simple decision tree for identifying ways in which WebSphere eXtreme Scale can be used to improve the scalability of an application.

2.4.1 WebSphere eXtreme Scale decision tree

It is assumed that while considering adopting WebSphere eXtreme Scale as a enterprise-wide object caching grid, IT professionals will engage in detailed systems and design analysis. This exercise is an important activity in devising a roadmap for adoption of any new technology. Depending on the issues at hand and the findings from the analysis, the decision tree shown in Figure 2-2 shows possible solutions based on WebSphere eXtreme Scale.

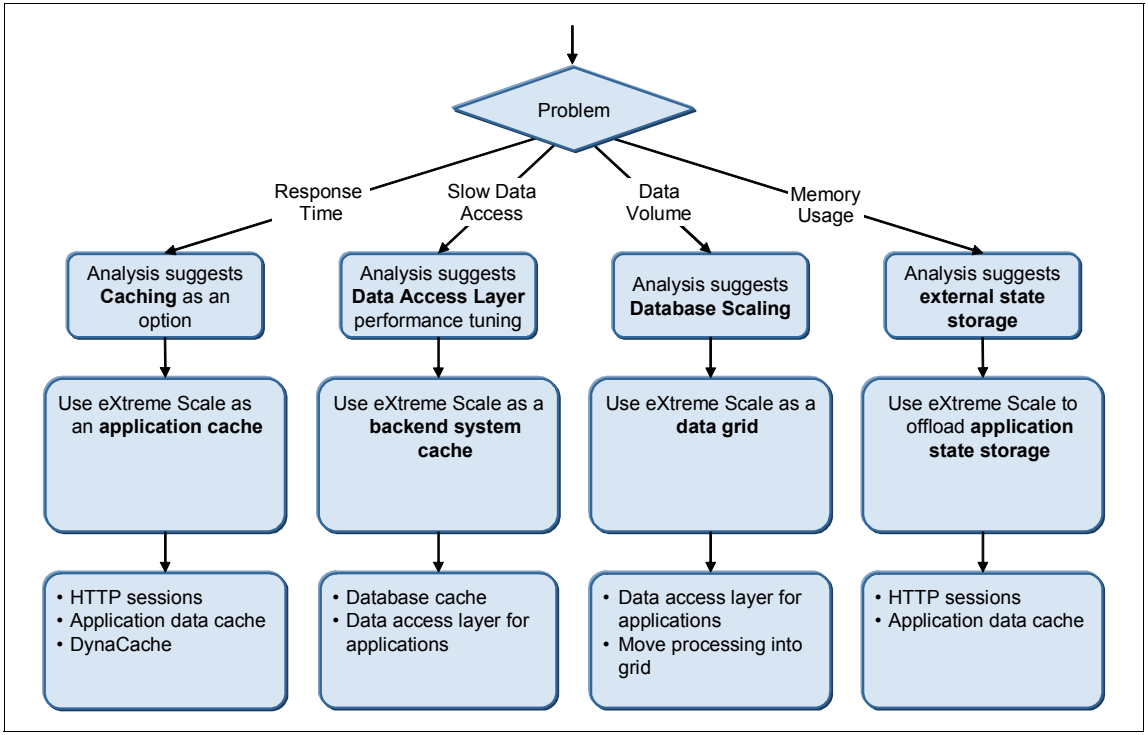


Figure 2-2 Decision tree for adopting WebSphere eXtreme Scale

The decision tree addresses the issues of inadequate response time, slow data access, data volume issues that affect performance, and memory usage problems. Based on analyses of similar problems, the first tier in the tree suggests the type of solution that would be appropriate at an architectural level. The next tier in the tree suggests how WebSphere eXtreme Scale can be used to implement the solution. The last tier provides information about the specific types of caching that might prove useful in resolving the problem.

The problem categories shown in the decision tree are interrelated, in that high database load will lead to slow data access, which itself is a cause of slow application response times. Removing one scalability bottleneck by introducing caching or a more advanced data grid solution will often reveal another bottleneck, so it may take a few iterations, and a few different uses of WebSphere eXtreme Scale, before the application meets its scalability targets.

The problems shown in the decision tree invite a number of incremental solutions, such as tuning database settings for greater performance, optimizing queries, and scaling up database servers. These do not address the greater scalability limitations of the application, but rather move it towards the limits of the current architecture. The effectiveness of these solutions depends on the inefficiency of the application or the availability of more powerful server equipment, both of which quickly reach points of diminishing returns.

The following sections provide more information about the options found at the bottom tier of the decision tree in Figure 2-2 on page 21.

2.4.2 HTTP sessions

For applications that make extensive use of HTTP session state, WebSphere eXtreme Scale provides an efficient coherent cache of session information across application servers, and introduces the possibility of failover between data centers for disaster recovery. Applications only need to be reconfigured to use WebSphere eXtreme Scale for HTTP session state storage.

See 3.3, “HTTP session state management” on page 28 for more information about this scenario.

2.4.3 Application data cache

WebSphere eXtreme Scale can be used to cache internal application data that is not directly represented in a database or other back-end system. This expands the amount of such data that can be cached by the application and improves the

cache hit rate by sharing the cache across application servers and potentially between related applications. This type of caching must be implemented within the application, requiring some development effort.

For more information about caching application data, see 3.7, “Application data caching” on page 36 and 3.4, “WebSphere dynamic cache service replacement” on page 31.

2.4.4 Database cache

WebSphere eXtreme Scale can be used as a scalable, shared, coherent cache for databases and other data sources, reducing the load on the back-end systems and improving the performance of the application. It offers drop-in caching solutions for the OpenJPA and Hibernate database persistence layers, meaning that only configuration changes are required for some applications.

For more information about using WebSphere eXtreme Scale as a cache for back-end systems such as database servers, see 3.5, “Database caching” on page 32.

2.4.5 Grid as data access layer

Using WebSphere eXtreme Scale as a data access layer allows access to its querying and indexing capabilities, which can reduce the load on back-end systems. This approach involves replacing the application’s existing data access layer using the WebSphere eXtreme Scale programming interfaces.

For more information about this approach, see 3.5, “Database caching” on page 32.

2.4.6 Moving application processing into the grid

Beyond caching and data access scenarios, WebSphere eXtreme Scale offers a programming model for processing data sets in parallel within the grid containers that hold the data. This model enables processing rates higher than those offered by relational database queries, and better scalability in terms of data size.

This approach requires significant application design effort to identify the operations to perform within the grid, and to implement those operations using the WebSphere eXtreme Scale programming interfaces.

For more information about using WebSphere eXtreme Scale in this way, see 3.8, “Application processing in the grid” on page 38.

3

Application scenarios

This chapter introduces the terminology used to describe data grids, and discusses the scenarios in which WebSphere eXtreme Scale can be employed to improve the scalability and availability of an application.

© Copyright IBM Corp. 2009. All rights reserved.

3.1 WebSphere eXtreme Scale terminology

This section discusses terms that are used to describe an eXtreme Scale installation. Figure 3-1 is an illustration of the terms discussed.

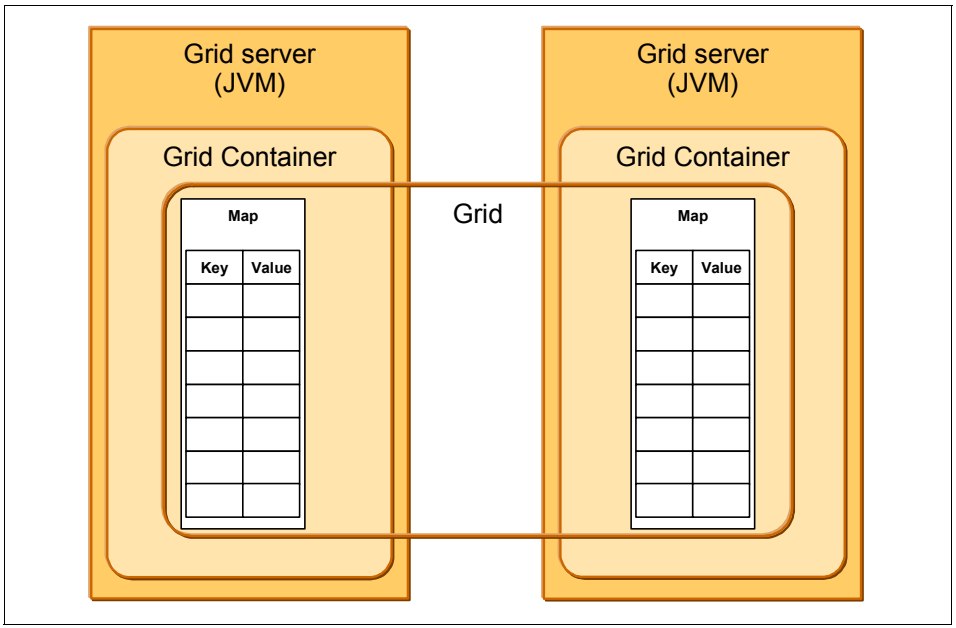


Figure 3-1 WebSphere eXtreme Scale components

- Map

  WebSphere eXtreme Scale stores data in maps. A map is an interface that stores data as key/value pairs. There are no duplicate keys in a map. A map is considered an associative data structure, because it associates an object with a key.

- Key

  A key is a Java object instance that identifies a single cache value. Keys can never change and must implement the equals() and hashCode() methods.

- Value

  A value is a Java object instance that contains the cache data. The value is identified by the key and can be of any type.

- Java Virtual machine

  A Java Virtual machine (JVM), is an execution environment that is platform independent. In the context of eXtreme Scale, a JVM can host one or more grid containers. A JVM can be either an application server or a stand-alone JVM.

- Grid servers

  The JVMs that host the catalog service (we will refer to these JVMs as *catalog servers*) and the JVMs that host grid containers holding the cache data are referred to as grid container servers. The catalog service manages the grid containers and provides a contact point for grid clients, while the grid container servers host the cache data.

- Grid containers

  Much like the typical containers in a JEE context, such as the Web and EJB containers, grid containers provide the grid application with services such as security, transaction support, and remote connectivity. The grid containers house partition distribution and placement, and enable easy manageability of the grid infrastructure. Much like other containers, the grid container can also take advantage of the configuration service provided by the WebSphere Application Server infrastructure in a managed environment.

- Grid

  Grids contain the maps that store data in WebSphere eXtreme Scale. Grids provide qualities of service such as scalability and replication of data.

- Partitions

  Partitioning is the process of splitting a grid into smaller sections such that the sections (partitions) may be scattered over available grid containers. Partitioning allows the grid to store more data than can be accommodated in a single JVM. The data is partitioned using an application-defined schema, and the number of partitions is specified by the application. Partitioning is an important consideration when designing and configuring a scalable infrastructure.

- Transactions

  Applications modify data cached by WebSphere eXtreme Scale using transactions, which have the same semantics as in a traditional database environment. A transaction holds an in-progress set of changes to the cache, which are all committed atomically to the cache, or are all discarded.

3.2 Application scenarios

The following sections present scenarios in which WebSphere eXtreme Scale can be employed to improve the scalability and availability of an application.

Most applications make some use of the user session state, caching, or database access facilities provided by the application server environment. WebSphere eXtreme Scale can enhance these facilities to improve the scalability and reliability of the application. In many cases, these improvements can be realized by installing WebSphere eXtreme Scale and updating the application configuration.

Beyond enhancing application server facilities, WebSphere eXtreme Scale can be used in a variety of ways within an application, ranging from caching long-lived application data to processing of large data sets in parallel.

3.3 HTTP session state management

Many Web applications use the HTTP session state management features provided by Java Enterprise Edition (JEE) application servers. Session state management allows the application to maintain state for the period of a user's interaction with the application. The traditional example for this is a shopping cart, where information about intended purchases is stored until the shopping session is completed.

To support high-availability and failover, the state information must be available to all application server JVMs. Application servers typically achieve this by storing the state information in a shared database, or by performing memory-to-memory replication between JVMs. When HTTP session state is stored in a shared database, scalability is limited by the database server. The disk I/O operations are an expensive performance cost to the application. When the transaction rate exceeds the capacity of the database server, the database server must be scaled up.

When HTTP session state is replicated in memory between application servers, the limits to scalability vary depending on the replication scheme. Commonly, a simple scheme is used in which each JVM holds a copy of all user session data, so the total amount of state information cannot exceed the available memory of any single JVM. Memory-to-memory replication schemes often trade consistency for performance, meaning that in cases of application server failure or user sessions being routed to multiple application servers, the user experience may be inconsistent and confusing.

WebSphere eXtreme Scale is an ideal platform to store HTTP session data. It provides non-invasive HTTP session state management in the form of a standard JEE HTTP servlet filter, as illustrated in Figure 3-2.

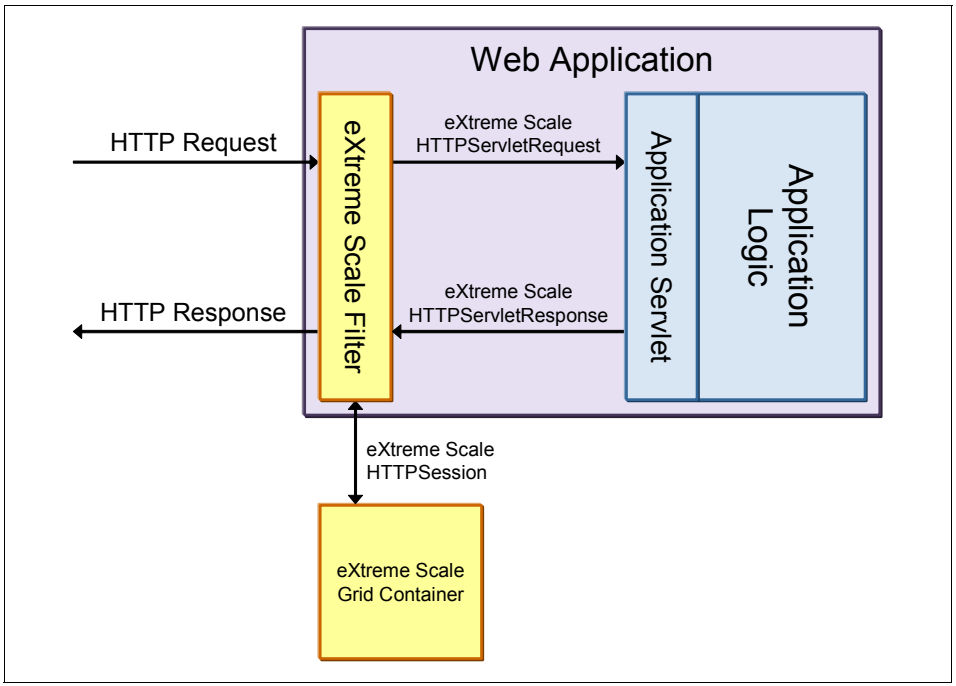


Figure 3-2 eXtreme Scale servlet filter

The servlet filter replaces the HTTP session state management facilities of the application server, instead fetching user session information from the data grid and writing changes back to the grid as necessary. Since the HTTP session data is transient in nature, it does not need to be backed to disk and can be contained completely in a highly-available replicated grid.

The grid is not constrained to any one application server product or to any particular management unit, such as WebSphere Network Deployment cells. User sessions can be shared between any set of application servers, even across data centers. This allows a more reliable and fault-tolerant user session state.

3.3.1 Benefits

Replacing a shared database or a memory-to-memory replication scheme with a shared in-memory cache for HTTP session state will improve the scalability of user session state. Removing the need to write the data to disk can also increase the performance of the application while providing a high quality of service for the session data.

The servlet filter works in any JEE-compliant application server and runtime environment. The servlet filter must be configured into the application, and some configuration details must be provided, but no application code changes are required.

Some application types and topologies can be supported much more effectively when using the eXtreme Scale session manager. For an application that has no server affinity (requests are sprayed across the server instances) the session state data can be stored in the grid and quickly available to any server as required.

Applications that are being migrated or must be supported on multivendor server environments (or different versions of the same application server) can also benefit by having a common store of session state data in the grid.

3.3.2 Limitations

This use of WebSphere eXtreme Scale only addresses one particular scalability concern that may affect any given application. This approach will be of limited benefit for applications that make little use of HTTP session state. Even for applications that use HTTP session state extensively, WebSphere eXtreme Scale may need to be employed in other ways to meet the scalability requirements for the application.

3.3.3 Topologies

Most commonly, the grid containers are collocated in the same application server JVMs that host the application itself. The session manager can communicate with the grid directly, thus avoiding network communications delays. When using WebSphere Application Server Network Deployment, the grid containers are automatically managed by the application server infrastructure.

The servlet filter may instead be configured to use a remote grid. This involves more configuration effort, and may provide slightly lower performance due to the overhead involved in accessing a remote grid, but it lowers the memory footprint for the application.

3.4 WebSphere dynamic cache service replacement

Many Web-based applications use dynamic page generation techniques, like JavaServer Pages (JSPs), for pages that rarely change, such as product details or information about corporate policies. Application servers generally provide at least an in-memory cache to store the generated output the first time the page is rendered, to save the processing work and back-end system load for subsequent requests.

WebSphere Application Server's dynamic cache service stores the output of servlets and JSPs, as well as custom application objects. It can be configured to off-load cached objects to disk, or to replicate cached objects across a cluster of application servers.

When the disk off-load option is enabled, the dynamic cache service extends its in-memory cache by writing cached objects to disk when the in-memory cache is full. If the working set of the application does not fit within the in-memory cache, disk access time becomes an important factor in application performance. Faster disk technologies, such as Storage Area Networks (SANs), are often used to improve performance.

When memory-to-memory cache replication is used, the size of the cache is limited to the available memory of any single JVM hosting the application. If the working set of the application outgrows the cache, the application will perform some amount of redundant work to re-create objects that have been evicted from the cache.

WebSphere eXtreme Scale provides a drop-in replacement for the dynamic cache service that stores cached objects in a grid, allowing the size of the cache to expand to the sum of all available memory in JVMs in the grid. The application's dynamic cache service policy remains in effect, so the carefully defined dependency and invalidation rules continue to function.

3.4.1 Benefits

Replacing a per-JVM disk off-loading cache with a single shared in-memory cache expands the maximum size of the cache, removes disk access speed as a factor in application performance, and enables incremental scaling of cache size that is not generally possible with SANs. The cache can be expanded by adding new grid containers at any time. As there is a single shared cache, each application server stores only a fraction of the cached data, reducing the memory footprint of the application.

A newly started application server no longer has to warm up its cache before reaching peak performance. The cached data is already available to it from the grid. As a result, the application's performance is more reliable, and scaling out the application does not result in load spikes at the back-end systems.

Memory usage for caching may be reduced by as much as 70% compared to per-server disk off-load caches, owing to the reduced redundancy of a single shared cache. Additionally, the eXtreme Scale dynamic cache service provider can compress the contents of the cache to save even more memory.

3.4.2 Limitations

This approach can only address the scalability limitations of the dynamic cache service. It may be necessary to combine this approach with others described in this chapter to reach the scalability goals for the application.

3.4.3 Topologies

The WebSphere eXtreme Scale dynamic cache provider is designed to work effectively with collocated or remote grid containers. Customers can choose to locate their cache data inside the applications servers that host the application, or inside of remote eXtreme Scale container processes. The remote container processes do not need access to customer application code unless custom key objects are being used to insert data through the dynamic cache service DistributedMap API.

3.5 Database caching

The back-end database is the scalability bottleneck for many database-centric applications. As the data volume and transaction rate expand, the application will eventually reach a point where the database server may not be scaled up any further.

WebSphere eXtreme Scale provides drop-in caching support for the OpenJPA and Hibernate data persistence frameworks. This support employs a grid as a second level cache in the JPA layer, as shown in the bottom of Figure 3-3 on page 33.

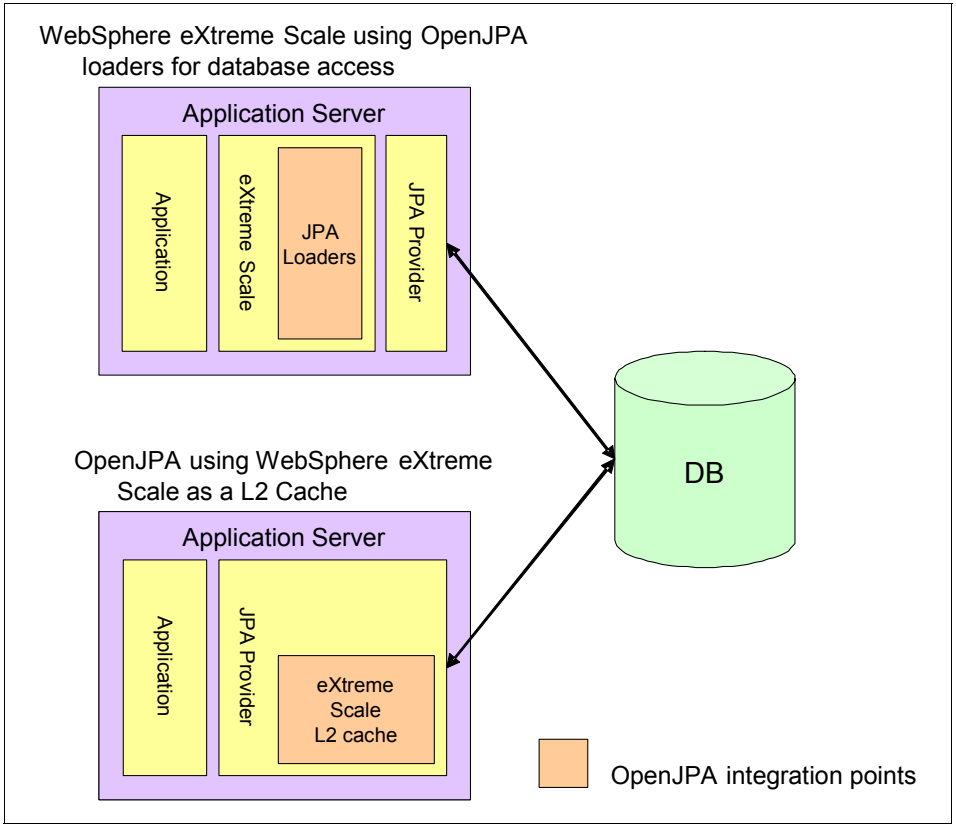


Figure 3-3 WebSphere eXtreme Scale database integration with JPA

A more intrusive approach, shown in the top of Figure 3-3, is to modify the application to use WebSphere eXtreme Scale interfaces to access the data. In this case, a grid acts as a cache or front-end shock absorber for the database. The database remains the system of record, but all data access from the application goes through the cache grid. The application would use the WebSphere eXtreme Scale programming interfaces to access the data.

In this variation, the application can either use the JPA-based database loader provided with WebSphere eXtreme Scale, or it can use a custom loader to retrieve and update grid entries from the back-end data store.

Using a custom database loader does not necessarily involve significant development effort. The application's existing data access code provides a starting point, and can potentially be reused as-is, leaving only the task of implementing the loader plug-in interface.

A customer employing this approach reduced the response time for their application from 60 ms to 6 ms. Previously, their database server had been saturated, limiting their application to 450,000 concurrent users and 80,000 requests per second. With WebSphere eXtreme Scale in place as a cache, reducing the database load, the application is able to scale beyond 1,000,000 concurrent users, enough to accommodate the customer's projected growth for the next two years.

3.5.1 Benefits

Using WebSphere eXtreme Scale to provide a cache for the database results in faster access to commonly read database entries, reduced load on database servers, and lower application latency.

The grid can also be configured to perform batch updates to the database for a period of time, providing write-behind caching. This improves the application response time since the database update is no longer part of the application transaction, and improves the efficiency of the database since it processes larger batched updates. Further, only the last of multiple changes in a batch must be written, and changes to multiple fields of one record are combined into a single write of the record.

3.5.2 Limitations

When using WebSphere eXtreme Scale as a write-behind cache, it is possible for the cache to become inconsistent with the database. This happens because the changes are saved in the cache for batching, and thus the database is out of date. This is much like the shadow database shown in Figure 1-1 on page 2 being out of date. This is only a problem if other applications are accessing the database and not going through the same eXtreme Scale data grid. See 4.11, "Handling of stale caches" on page 74 for ways to deal with stale data.

With write-behind caching, if a delayed write fails due to database failure or connectivity issues, the failed database updates are saved in a separate map and applied to the database when the database or connection is once again available. If the database is up but the delayed write fails, the grid will evict the entry that failed to write to the database, but it has no way to notify the application directly since the application's transaction has already been committed.

3.5.3 Topologies

The grid can be hosted in a separate set of servers in order to decouple grid availability from application availability, and to maximize the memory available for caching data.

3.6 Caching for other back-end systems

While a relational database system is an obvious candidate for caching, especially in cases where the database is used for persistent storage of Java objects, caching applies to other types of back-end systems also.

Depending on the nature of the systems involved, some proportion of the work performed by the back-end system may be redundant. Generally the redundant work will be repeated requests for rarely updated information. Careful analysis is required to determine which requests are cacheable and which must be performed by the back-end system each time.

If the back-end system is accessed through an Enterprise Service Bus (ESB), it is possible to introduce caching as a mediation on the bus without modifying the application, but otherwise, the cache access would need to be implemented within the application.

After performing a detailed analysis of back-end system requests from an application, one customer reported that over a 24 hour period, 25% of all requests for user profiles were redundant and could be cached. To realize any benefit from this, a request cache would need to be large enough to store all responses from that 24 hour period, and would need to be shared across all application servers. A simple in-memory cache within the application would not provide a high enough cache hit rate to make a significant difference.

After inserting a cache for user profiles, the logon time for users with cached profiles was 20 ms, down from 700 ms without the cache. The reduced load on the mainframe system holding the user profiles saves the customer $500,000 per month.

3.6.1 Benefits

As with the database scenarios described above, this form of caching reduces the load on the back-end systems, freeing up capacity to perform new tasks or to handle increasing application load.

3.6.2 Limitations

If an application makes few cacheable requests to back-end servers, then there is little to be gained by introducing caching. In this context, cacheable means the same data is retrieved multiple times within a relatively short time period.

3.6.3 Topologies

In this scenario, the application may require a large cache to provide a useful cache hit rate. In order to provide the largest possible cache, the cache grid is generally hosted on a separate set of servers.

3.7 Application data caching

In some cases, slow application response times cannot be attributed to any single cause, but instead are caused by the cumulative cost of accessing back-end services and performing processing tasks throughout the application.

In this situation, the way to reduce response time is to identify redundant back-end operations in the application and then insert a cache to store the results of these operations. Before performing each of these cacheable operations, the application constructs a cache key corresponding to the input of the operation and uses it to see if the result has already been cached. If not, the application performs the operation and caches the result.

The main difference between this approach and those described in 3.5, “Database caching” on page 32 is that the cache does not act as a transparent layer in front of the database, and it is possible to cache transient data calculated within the application or created by combining the results of multiple back-end operations.

Depending on the complexity of the application and its back-end data sources, it may require significant time and effort to identify the operations that are cacheable, and integrating the cache into the application may require changes throughout the application code, rather than isolated at a particular layer as with previously described approaches. If the application already has some kind of caching logic, WebSphere eXtreme Scale can easily be integrated. Figure 3-4 on page 37 shows the cache integrated into a typical application architecture.

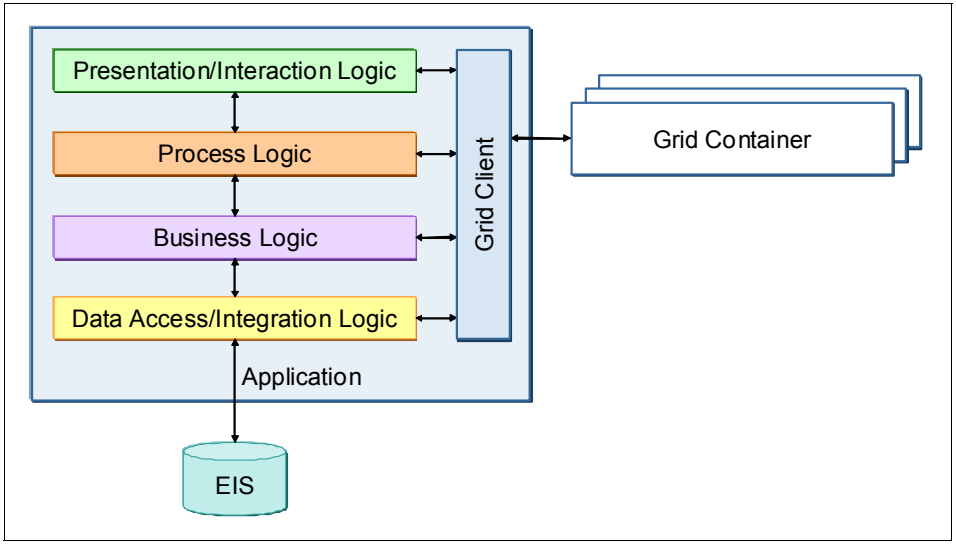


Figure 3-4 Cache scenario integrated into generic application architecture

An enterprise employed this approach to cache user-specific information retrieved from multiple back-end servers. The application built a single XML document containing all of the information from the different data sources, so rather than cache each data source, the customer elected to cache the combined document. In addition to reducing the load on the back-end servers, this also improves the efficiency of the application servers, as they spend less time combining data to produce the XML documents.

3.7.1 Benefits

Caching commonly accessed data reduces the application response time, improving the user experience, and reduces the load on the back-end systems. As the cache request rate and data volume grows, the cache can be incrementally scaled by adding more grid containers.

The primary benefit of using WebSphere eXtreme Scale in this scenario, as compared to other forms of cache, is that the cache contains less redundancy. An object is stored only once in the cache, even if multiple clients use it. Thus, the same amount of memory used for caching will store more data, which increases the cache hit rate.

Additionally, a WebSphere eXtreme Scale cache may be shared between multiple applications. If a set of related applications use a common representation of a user profile, for example, the profile can be cached once by the first application that the user access, and then accessed quickly by all the others.

3.7.2 Limitations

In this caching scenario, the cached data can only be accessed by primary key. The cache is not fully populated, so querying the cache would only produce partial results, dependent on recent cache accesses.

Care must be taken to configure the cache such that data remains in the cache for long enough to be useful, but the cache does not grow too large.

When the grid container is not co-located with the application, WebSphere eXtreme Scale will create near caches where possible, as determined by the grid configuration. Special attention has to be paid to handle stale data in these near caches. See 4.11, “Handling of stale caches” on page 74.

3.7.3 Topologies

Typically, multiple stand-alone JVMs are used to host the grid containers because the grid does not require special infrastructure, but as much memory as possible.

3.8 Application processing in the grid

Inserting a fully populated, partitioned, in-memory cache into the application architecture introduces the possibility of processing the entire data set in parallel at local memory access speeds. Rather than fetching data from the back-end data store and processing it in series, the application can process the data in parallel across multiple grid containers and combine the results together. In general, this relies on the data items being independent of each other. In cases where the data items are interrelated, the cache must be partitioned such that all related items are placed within the same partition.

In this scenario, the grid usually becomes the system of record. The database remains as a hardened data store to ensure against data loss caused by catastrophic failure of the systems hosting the grid.

Applying this technique involves changes to the application logic since the data processing operations must be specified in terms of the parallel processing schemes provided by the grid. As shown in Figure 3-5, the grid becomes a central component of the application architecture, rather than a solution applied at individual points within the application.

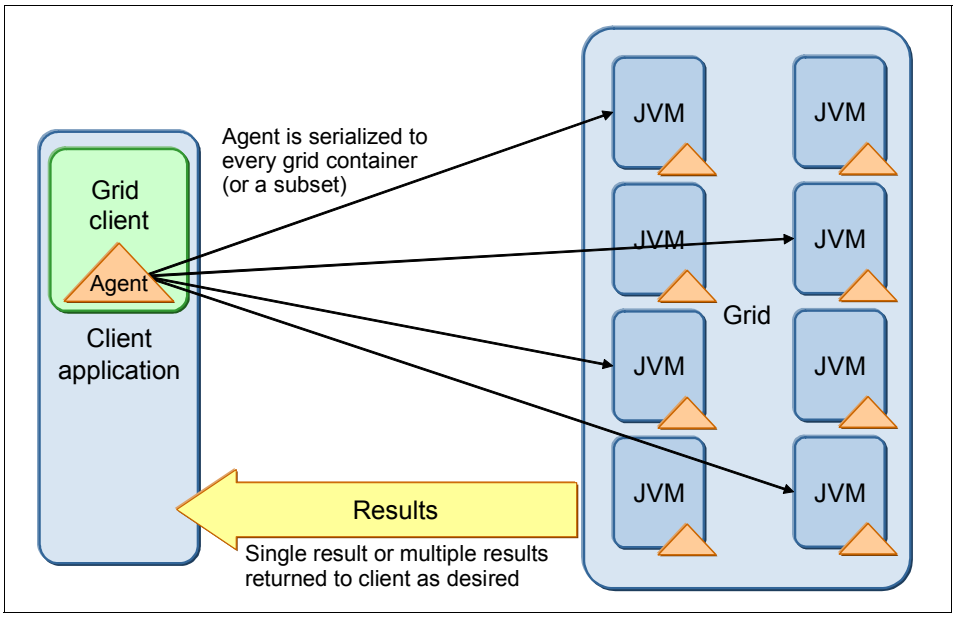


Figure 3-5 Agents sent out to the grid to collocate themselves with the data

An agent is code created using the DataGrid APIs, which provide a vehicle to run logic collocated with the data, to query across partitions, and to aggregate large result sets into a single result set. That is, the agent does the collation of the data into a single unit and returns it to the client.

Two major query patterns exist for DataGrid applications:

- Parallel map

  In this pattern, the entries for a set of entities or objects is processed and a result for each entry processed is returned.

- Parallel reduction

  In this pattern a subset of the entries is processed and a single result is calculated for the group of entries.

It is not necessary to replace all application data access with grid agents. The WebSphere eXtreme Scale APIs used to implement the other approaches in this chapter are still available for operations that do not involve accessing large amounts of data from the grid, or that cannot be expressed in terms of the grid agent query patterns described above.

3.8.1 Benefits

This approach enables parallel processing of large volumes of data in ways that most likely were not practical without the grid. It may be possible to run processes online that previously could only be run as batch jobs.

An application using grid agents can be scaled out in terms of both data size and processing capacity by adding more grid containers.

3.8.2 Limitations

Unlike other approaches, in which the grid is used as an adjunct to an existing data store, using grid agents involves redesigning some aspects of the application.

When using grid agents, the effects of partitioning the data in the grid become more apparent. When run in parallel across multiple grid containers, each instance of the grid agent can only access data from a single partition. If the agent needs to compare or combine multiple data items, the data partitioning scheme must be designed to place the related data items in the same partition.

Because grid agents run within the grid container process, the business logic must also be deployed into the grid container. Other approaches may only require the data cache objects.

3.8.3 Topologies

Due to the large amounts of data typically stored in grid applications of this type, the grid containers are not usually collocated with the application.

4

Architecture, design concepts, and topologies

This chapter provides an overview of the architecture of the WebSphere eXtreme Scale product. It explains the terminology for the main components and how these components provide grids for data storage. It discusses configuration and management issues, and common topologies. It also covers the programming aspects including development environments and APIs used to access the grid.

© Copyright IBM Corp. 2009. All rights reserved.

4.1 WebSphere eXtreme Scale architecture

This section discusses the basic concepts required to understand how WebSphere eXtreme Scale is structured and how applications use it.

4.1.1 User view

Figure 4-1 gives a logical user view of a grid. A user connects to a grid, then accesses the maps in that grid. Data is stored as key value pairs in the maps. Figure 4-1 shows grid A, which has two maps: A and B.

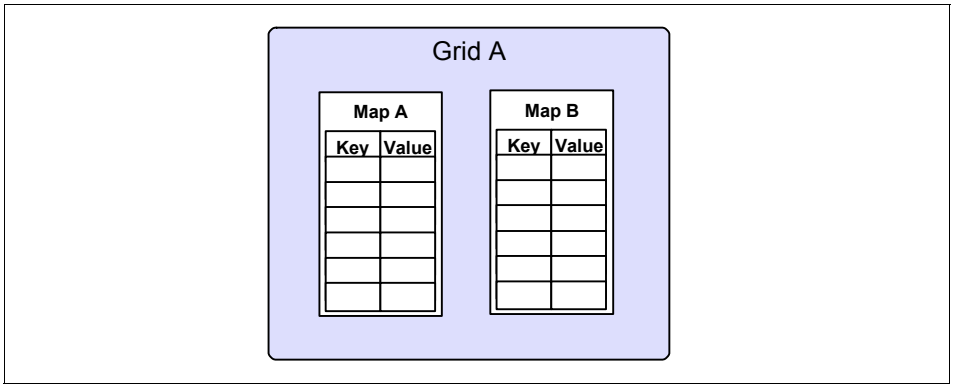


Figure 4-1 User view of a WebSphere eXtreme Scale grid

We have been loose with the term “grid”. We have used it to refer to the entire WebSphere eXtreme Scale infrastructure, or that part that stores and manages your data. However, the term grid has a precise technical definition in eXtreme Scale, namely, a container for maps that contain the grid’s data. Clients connect to grids, and access the data in the map sets they contain.

Figure 4-2 on page 43 exposes the next level of detail by introducing the map set and partitioning concepts. A map set is a collection of, or container for, maps. So a grid is really a collection of map sets. The key abstraction introduced by a map set is that it may be partitioned. This means it may be split into parts that may be spread over multiple containers.

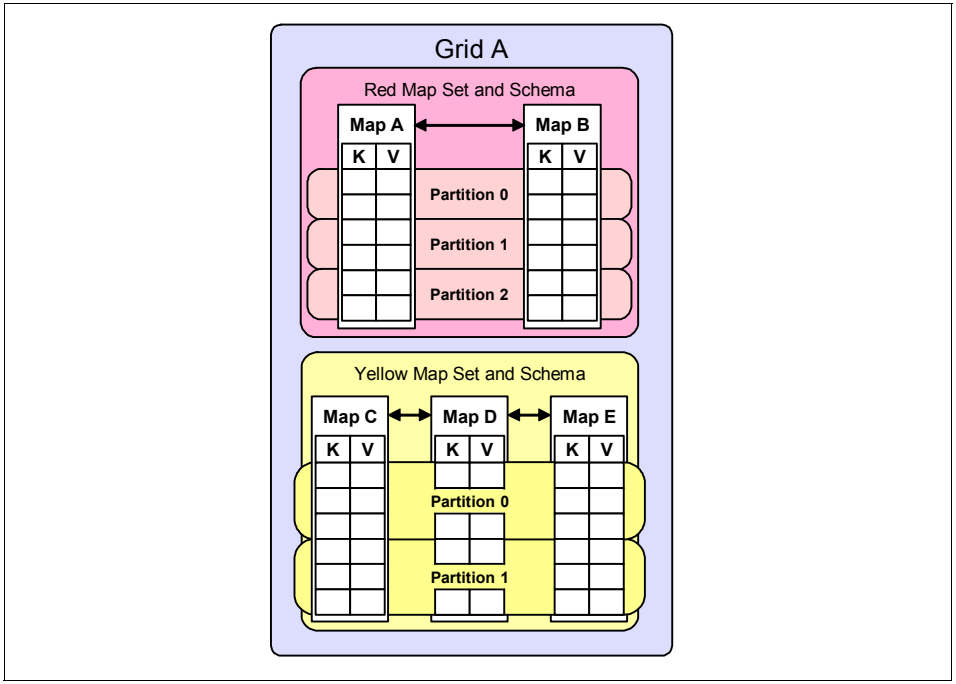


Figure 4-2 Maps in partitioned map sets in a grid

Now we can provide a more detailed definition of our terms.

- Map

  A map is an interface that stores data as key-value pairs (duplicate keys are not supported). A map is considered an associative data structure, because it associates an object with a key. The key is the primary means of access to the data in the grid. The value part of the map is the data typically stored as a Java object.

- Key

  A key is a Java object instance that identifies a single cache value. Keys can never change and must implement the equals() and hashCode() methods.

- Value

  A value is a Java object instance that contains the cache data. The value is identified by the key and can be of any type.

- Map set

  A map set is a collection of maps whose entries are logically related. More formally, a map set is a collection of maps that share a common partitioning scheme. Key elements of this scheme are the number of partitions, and the number of synchronous and asynchronous replicas.

- Grid

  A grid is a collection of map sets.

- Partitions

  Partitioning is the concept of splitting data into smaller sections. Partitioning allows the grid to store more data than can be accommodated in a single JVM. It is the fundamental concept that allows linear scaling. Partitioning happens at the map set level. The number of partitions is specified when the map set is defined. Data in the maps is striped across the N partitions using the key hashcode modulo N. Choosing the number of partitions for your data is an important consideration when configuring and designing for a scalable infrastructure.

  Figure 4-2 on page 43 shows two map sets in a grid: Red and Yellow. The Red map set has three partitions, and the Yellow map set has two partitions.

- Shards

  Partitions are logical concepts. The data in a partition is physically stored in shards. A partition always has a primary shard. If replicas are defined for a map set, each partition will also have that number of replica shards. Replicas provide availability for the data in the grid.

- Grid containers

  Grid containers host shards, and thus provide the physical storage for the grid data.

- Grid container server

  A grid container server hosts grid containers. It is WebSphere eXtreme Scale code running either in an application server or a stand-alone JSE JVM. Often grid container servers are referred to simply as JVMs.

Figure 4-3 on page 45 shows how these concepts are related.

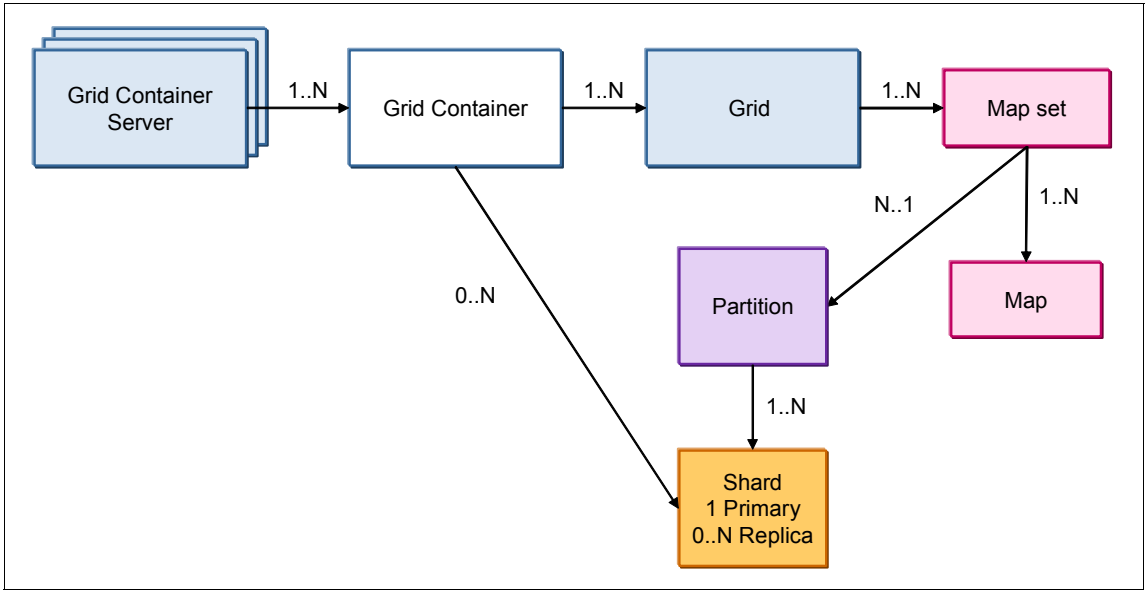


Figure 4-3 WebSphere eXtreme Scale meta model

- A grid container server can host many grid containers.
- A grid container hosts shards, which may be from one or more grids. A grid can be spread across many grid containers. The catalog service places shards in grid containers.
- A grid consists of a number of map sets. A map set is partitioned using a key. Each map in the map set is defined by a BackingMap.
- A map set is a collection of maps that are typically used together. Many map sets can exist in one grid.
- A map holds (grid) data as key value pairs.
- A map set consists of a number of partitions. Each partition has a primary shard and *N* replica shards.

The example shown in Figure 4-4 shows the parts and concepts we have discussed.

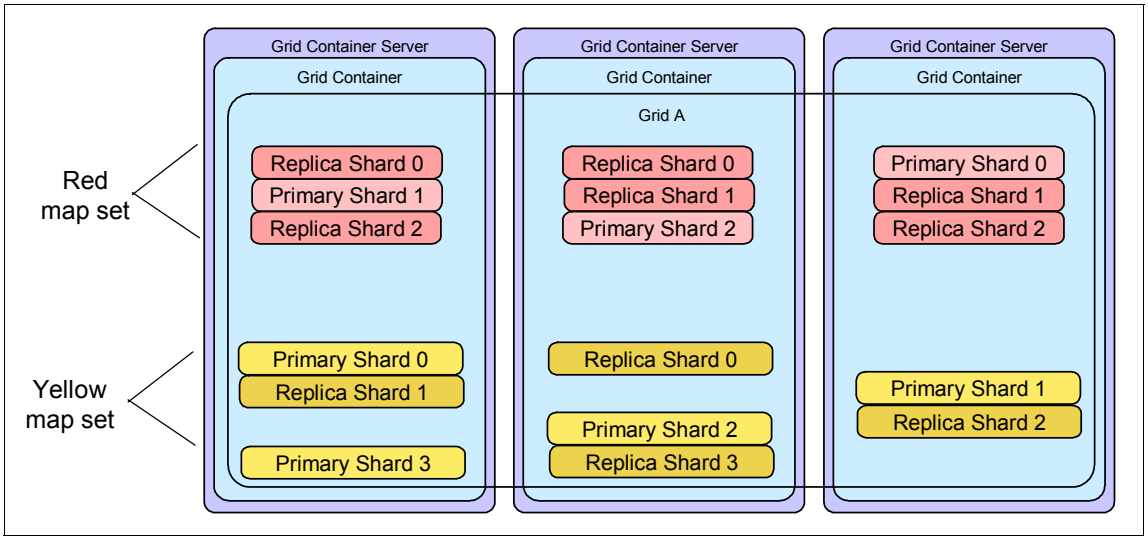


Figure 4-4 Grid A is stored physically in shards

Figure 4-4 shows an example of how grid A can be physically stored in WebSphere eXtreme Scale. Grid A contains two map sets: Red and Yellow. The Red map set has three partitions and two replicas, for a total of nine shards. (nine comes from three partitions times three shards for each partition, the primary plus two replicas.) The Yellow map set has four partitions and one replica, for a total of eight shards. The figure shows one way the shards could be distributed over the available grid containers.

4.1.2 Shards

The catalog service is the real brains for WebSphere eXtreme Scale. It places shards in grid containers. The shard distribution algorithms ensure that the primary and replica shards are never in the same container (or in a container with the same IP address) to ensure fault tolerance and high availability. If the machine or grid container hosting the primary shard fails, the catalog service promotes a replica shard to be the primary shard, and creates a new replica shard in another grid container on another IP address (usually a separate machine). The replica shard is then populated by copying the data from the new primary. If a machine or grid container hosting a replica shard fails, the catalog service creates a new replica in another grid container on another IP address, and is then populated by copying the data from the primary.

There are three types of shards:

- Primary
  - Handles all insert, update and remove requests. (Replicas are read-only.)
  - Replicates data (and changes) to the replicas.
  - Manages commits and rollbacks of transactions.
  - Interacts with a back-end data store for read and write requests.
- Synchronous replica
  - Maintains the exact state as the primary shard.
  - Receives updates from the primary shard as part of the transaction commit to ensure consistency.
  - Can be promoted to be the primary in the event the primary fails.
  - Can handle get and query requests if configured to do so.
- Asynchronous replica
  - May or may not have exactly the same state as the primary shard.
  - Receives updates from the primary after the transaction commits. The primary does not wait for the asynchronous replica to commit.
  - Can be promoted to be a synchronous replica or primary in the event of the primary or a synchronous replica failure.
  - Can handle get and query requests if configured to do so.

Note: A primary shard is sometimes referred to as the primary partition. While you may see those two terms used interchangeably, a partition is composed of a primary shard and zero or more replica shards.

As the grid membership changes and new container servers are added to accommodate growth, the catalog service pulls shards from relatively overloaded containers and moves them to a new empty container. With this behavior, the grid can scale out, by simply adding additional grid container servers. Conversely, when the grid membership changes due to the failure or planned removal of grid container servers, the catalog service will attempt to redistribute the shards that best fit the available grid container servers. In such a case, the grid is said to scale in. The ability of WebSphere eXtreme Scale to scale in and scale out provides tremendous flexibility to the changing nature of infrastructure.

4.2 Catalog service

The catalog service, as shown in Figure 4-5, is a highly available service that maintains the healthy operation of grid servers and containers. The catalog service is made highly available by starting more than one catalog service process. The catalog service becomes the central nervous system of the grid operation by providing the following essential operation services:

- Location service to all grid clients
- Health management of the catalog and grid container servers
- Shard distribution and re-distribution
- Policy and rule enforcement

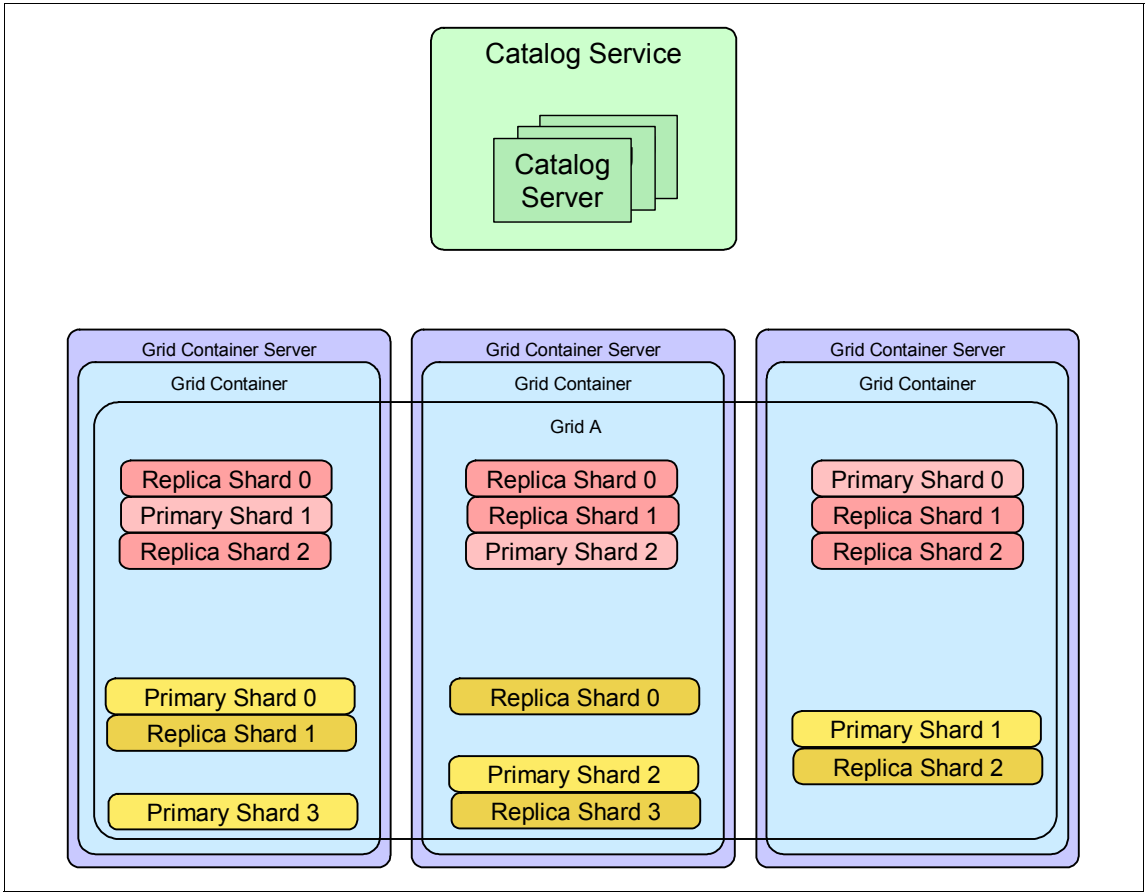


Figure 4-5 WebSphere eXtreme Scale grid

4.2.1 A WebSphere eXtreme Scale grid

Previously, we defined a grid as an object that holds maps in map sets, and that clients connect to access their data in the grid. We are now going to use the term grid to define an eXtreme Scale grid (Figure 4-5 on page 48). In eXtreme Scale, there are two kinds of severs: catalog servers and (grid) container servers. Catalog servers are the JVMs that comprise the catalog service. Container servers hold the shards, which is the data contained in the whole eXtreme Scale grid. Multiple catalog servers are recommended for the catalog service for availability of the service.

A catalog server uses a hostname and two ports to communicate with its peer catalog servers to provide the catalog service for the grid. This set of hostnames and port numbers from all the catalog servers defines the catalog service and are the grid's catalog service endpoints. Each catalog server must be given the complete set of catalog service endpoints when it starts. The catalog servers communicate among themselves on these ports to provide a highly available catalog service for the grid.

Each catalog server listens on a separate port for container servers and grid clients. This hostname and port on which a catalog server listens is also called a catalog server endpoint. You have to keep the context straight. Usually, you deal with container servers and clients, so this is the predominant sense of the term.

The net here is that when a container server starts, it is given its catalog server endpoint, by which it will connect to the catalog service.

The good news is that an eXtreme Scale grid is defined by the catalog service's catalog service endpoints. Container servers and clients connect to the grid by using a catalog server's client catalog service endpoint.

Container servers connect to the catalog service when they start. The catalog service thus knows the identity of all its container servers, and distributes the shards over these containers.

4.2.2 Shard placement

The catalog service plays an instrumental role in the elastic nature of the grid configuration. It is responsible for replication, distribution, and assignment of the shards to grid containers. The catalog service evenly distributes the primary shards and their replicas among the registered container servers.

Water flow algorithm

The mechanism employed to distribute shards among the available grid containers is based on an algorithm resembling the natural flow of water. As grid container servers leave and join the grid, shards are re-distributed.

This re-distribution maintains shard placement rules, such as not placing primary and replica shards in the same container (or even the same machine, to maintain high availability). 4.4, “Zones” on page 58 introduces another important shard placement rule.

It is important to understand the implications of the shard placement policy defined and enforced by the catalog service. The water flow algorithm ensures the equitable distribution of the total number of shards across the total number of available containers. Hence, WebSphere eXtreme Scale ensures that no one container is overloaded when other containers are available to host shards. eXtreme Scale also enables fault tolerance when the primary shard disappears (due to container failure or crash) by promoting a replica shard to primary shard. If a replica shard disappears, another is created and placed on an appropriate container, preserving the placement rules. Other key aspects of the approach are that it minimizes the number of shards moved as well as the time required to calculate the shard distribution changes. Both of these concerns are important to allowing linear scaling.

To ensure high (or continuous) availability of a data partition, WebSphere eXtreme Scale ensures that primary and replica shards of a partitions are never placed in the same container or even on the same machine.

Figure 4-6 on page 51 shows four partitions, each with a primary shard and one replica shard, distributed across four containers.

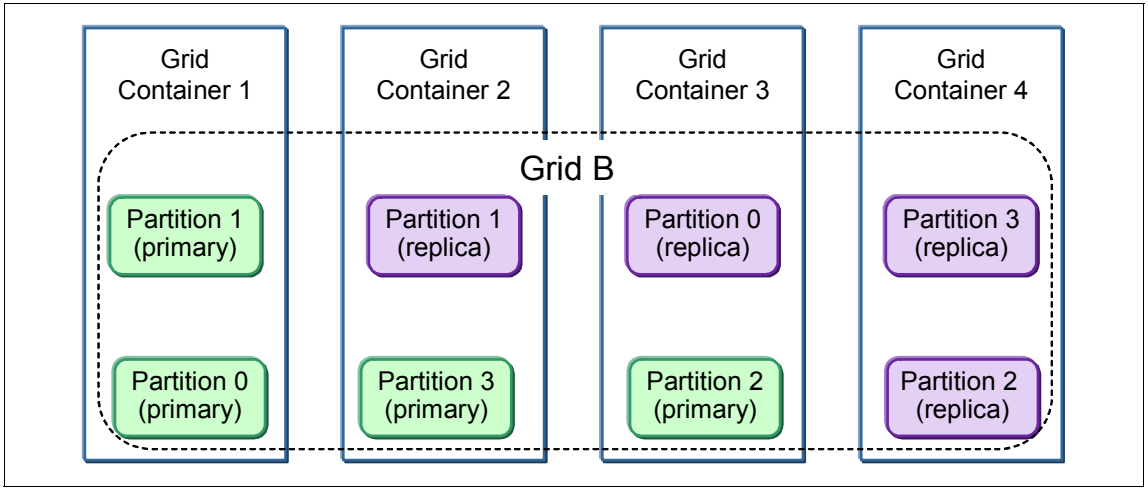


Figure 4-6 Shards placed on all available containers

Figure 4-7 shows how the shard placement would adjust if one of the containers failed and only three containers were available for placement. In this case, no primary partitions were affected so no replicas were promoted to be primary partitions. The two failed replica partitions are simply recreated in another container in the grid.

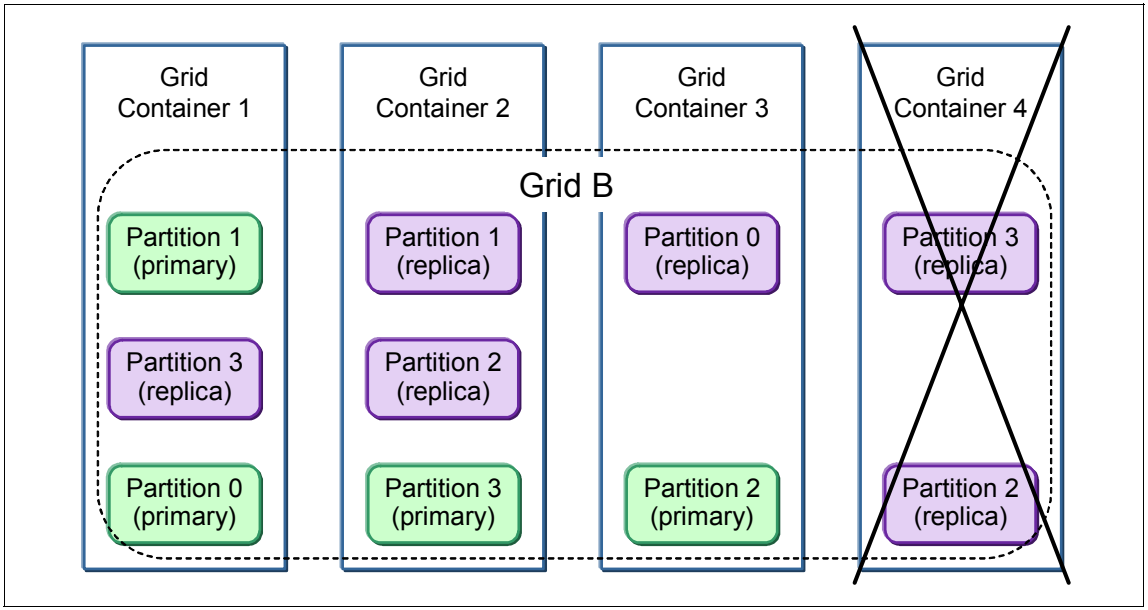


Figure 4-7 Shard placement and re-distribution after Grid Container 4 JVM Failure

4.2.3 Client grid access

Users access data in a grid through the WebSphere eXtreme Scale grid client, which is WebSphere eXtreme Scale code packaged in an ogclient JAR file. The primary user input is the catalog service endpoint, and grid and map names. When we talk about the grid client in this section, we are referring to the function that this WebSphere eXtreme Scale code performs. The user writes no such code.

The grid client begins its access to the grid by obtaining a routing table from the catalog service. Given an object to access, the client calculates the key's hashcode, divides that by the number of partitions, and the remainder is the number of the partition containing the object. The routing table enables the client to locate the partition's primary shard, which contains the desired object. In the event of a container failure, or re-distribution of partitions due to a change in grid container membership, the client is provided with an up-to-date routing table by one of the grid container servers.

When a client is unable to get a response from any of the containers hosting a shard from its route table, the client will contact the catalog service again. If the catalog service is not available, the client fails.

Clients have minimal contact with the catalog service, and network traffic is minimized when shards move, both of which enhance linear scalability.

4.2.4 Catalog service availability

Since the catalog service's role as the central nervous system is critical, it is necessary to consider the availability of the catalog service. The catalog service controls shard placement and routing for all clients. The catalog service should be clustered for high availability. This must be taken into account during the planning phases of the grid topology.

A catalog server can be configured and run in any JVM in the environment. A catalog server can be started as a stand-alone JVM or configured to be embedded into a WebSphere application server infrastructure process.

The catalog servers communicate together and typically rely on maintaining a quorum to determine the changes that need to be made to the grid configuration. Unless overridden, quorum is the full set of catalog servers. The catalog service will only respond to container events while the catalog service has quorum. Using this mechanism ensures that the grid configuration is not corrupted in the event of a catalog server loss due to a JVM or network failure. Intentionally stopping a catalog server instance does not cause loss of quorum.

When quorum is lost due to a temporary outage, it is re-established when the catalog server comes back online. When quorum is lost due to a more permanent or lengthy outage, the administrator can override quorum.

For more information, see *Catalog server quorums* at:

http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r0/topic/com.ibm.websphere.extremescale.over.doc/cxsquorcatsr.html

Note: Designing high availability grids requires careful planning, sizing, and configuration. More details on high availability and failure modes can be found in the *Availability* topic of the WebSphere eXtreme Scale information center. This topic is available at the following Web page:

http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r0/topic/com.ibm.websphere.extremescale.over.doc/cxsavail.html

4.3 WebSphere eXtreme Scale internal components

The following sections define the components you will find in WebSphere eXtreme Scale. Figure 4-8 is an illustration of the terms discussed.

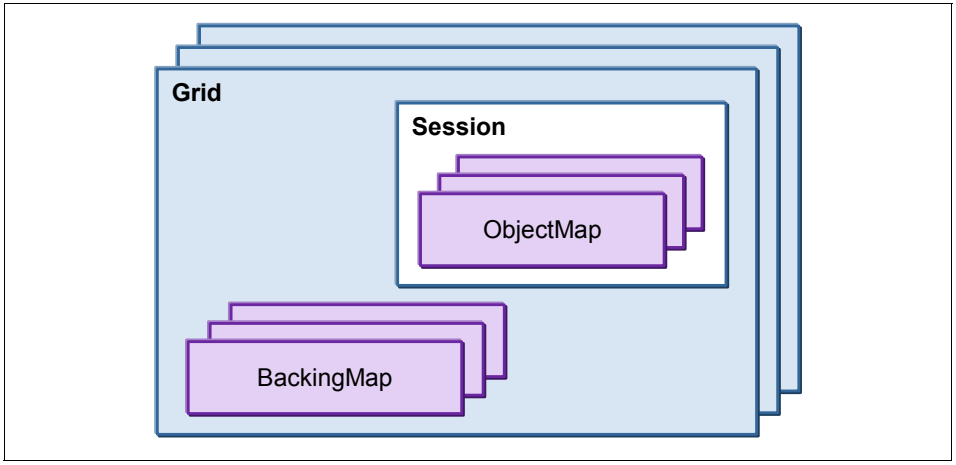


Figure 4-8 WebSphere eXtreme Scale internal components

4.3.1 Session

When a user initially interacts with a grid (the kind that has maps with data in map sets), a session is established. Applications begin and end transactions using the session interface. Sessions are not concurrently shared by multiple threads. When another user accesses the grid, another session is established.

4.3.2 Map

A map is an interface that stores data as key/value pairs. There are no duplicate keys in a map. A map is considered an associative data structure, because it associates an object (the value) with a key.

4.3.3 ObjectMap

An ObjectMap is a type of map that is used to store a value for a key. That value can be either an object instance or a tuple.

- An object instance requires its corresponding class file to be in the container JVM, because the bytecode is needed to resolve the object class.
- A tuple represents the attributes of an object. You do not need the class file present.

An ObjectMap is always located on a client, and is used in the context of a local session.

Figure 4-9 on page 55 illustrates three different ObjectMaps. An ObjectMap holds key objects and value objects. From left to right, the first ObjectMap contains an integer key and a string value. The next ObjectMap contains a (hex) integer key and a compound value. The last ObjectMap contains a compound key and compound value.

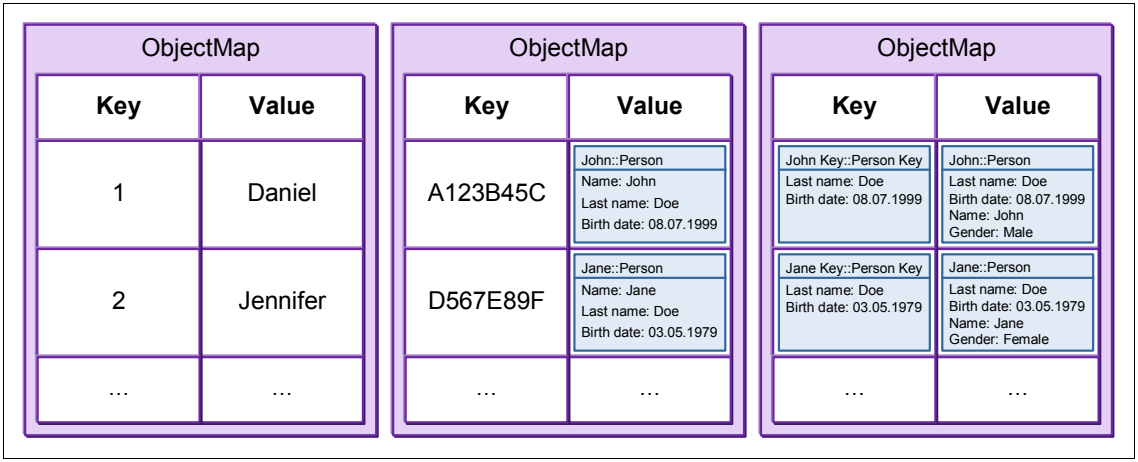


Figure 4-9 ObjectMap examples

4.3.4 Tuples

A tuple is an object used to represent compound objects. A tuple is simply an array of primitive types. It contains information about the attributes and associations of an entity. The EntityManager converts each entity object into a key tuple and a custom value tuple representing the entities (Figure 4-10). This key/value pair is then stored in the entity's associated ObjectMap.

The true value of tuples is in not having to define classes when several objects (entities) are represented or packaged together. When a loader is used with the map, the loader will interact with the tuples.

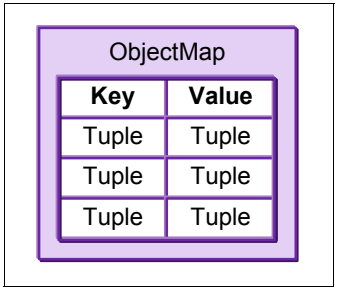


Figure 4-10 Tuples

4.3.5 Backing maps

A backing map, represented by a BackingMap object, contains cached objects that have been stored in the grid. An ObjectMap and a BackingMap are related through a grid session. The session interface is used to begin a transaction and to obtain an ObjectMap, which is required for performing transactional interactions between an application and a BackingMap object.

ObjectMaps and BackingMaps can reside in the same JVM that is hosting the local grid. (See 4.10.4, “Collocated application and cache topology” on page 71 for more details) BackingMaps can also reside in a container JVM separate from the ObjectMaps and the two maps will communicate remotely to persist data.

Each entity has its own BackingMap. Any serializable entity attributes are persisted to the BackingMap. This implies that each BackingMap has its own loader instance. The BackingMap will request any needed data that it does not contain from its loader, which in turn, will retrieve it from the backing store. This process is illustrated in Figure 4-11 on page 57. Note that, as shown in the figure, loaders work on primary shards only, not replicas.

Note: A loader is a plug-in to the BackingMap. The loader is invoked when data is requested from the grid and that data is not already in memory or when data is changed in the grid. The loader has the logic necessary for reading and writing data to the backing store. This is applicable in a write-through and write-behind type of topology where the application’s interactions are directly to the grid and it does not access the backing store itself. Built-in JPA loaders are provided to simplify this interaction with relational database back-ends. The JPA loaders use Java Persistence Architecture (JPA) APIs to read and write the data to the database back-ends. Users provide a `persistence.xml` file to plug in the JPA provider, such as OpenJPA or Hibernate. A service provider interface is also provided to support other back-end data stores.

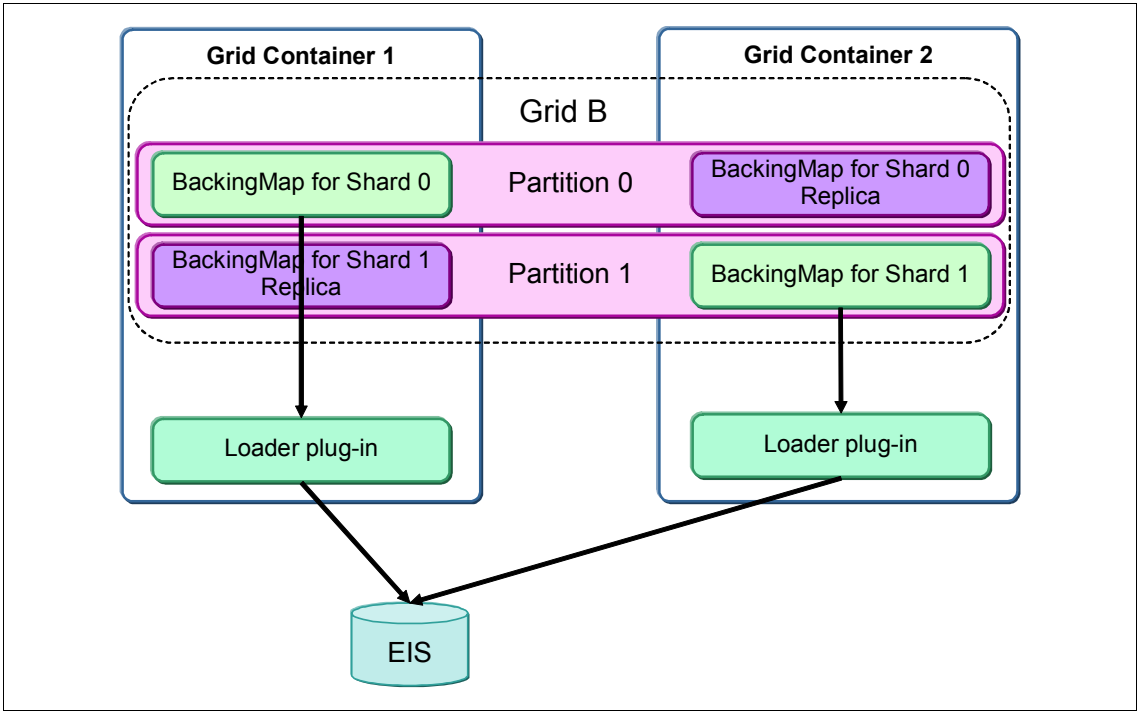


Figure 4-11 Example use of a BackingMap with a Loader

4.3.6 Grid clients and backing maps

The following terms are used when discussing how an application interacts with the grid.

- Grid client

  A grid client is WebSphere eXtreme Scale code from the `ogclient.jar` file that interacts with grid servers on behalf of application code. Grid clients connect to a grid through the catalog service. Grid clients examine the key of the application data to route the request to the proper partition. A grid client contains an ObjectMap and may contain a near-cache copy of a BackingMap.

  A grid client and grid server can have independent BackingMaps (near-cache and far-cache). The server-side, or far-cache, BackingMap is always shared between clients, while the client-side, or near-cache, BackingMap (if in use) is shared between all threads of the grid client. Clients can read data from multiple partitions in a single transaction. However, clients can only update a single partition in a transaction.

- ObjectGrid Instance

  Applications must obtain an ObjectGrid instance to work with a grid. This is done so that the application can interact with the grid and perform various operations, such as create, retrieve, update, and delete the objects in the grid.

4.4 Zones

Zone support provides much needed control of shard placement in the grid. Zone support is a significant competitive differentiator in the in-memory data grid (IMDG) space.

Zone support allows for rules-based shard placement, enabling high availability of the grid due to the placement of replica shards across physical locations. This notion is particularly appealing to enterprise environments that need data replication and availability across geographically-dispersed data centers. In the past, these enterprise computing environments were limited due to the performance constraints imposed by networks and real-time data replication requirements. With the inclusion of zone support, WebSphere eXtreme Scale offers better scalability by decoupling the grid environment. With zone support, only the metadata shared by the catalog servers is synchronously replicated, while the data objects are copied asynchronously across networks. This not only enables better location awareness and access of objects, but also imposes fewer burdens on enterprise networks by eliminating the requirement of real time replication.

As long as the catalog service sees zones being registered (as the zoned grid container servers come alive), the primary and replica shards are striped across zones. Further, the zone rules described in the grid deployment descriptor file will dictate placement of synchronous or asynchronous replica shards in respective zones.

As a general practice, it is recommended that you place only synchronous replicas in same zone and asynchronous replicas in a different zone for optimal replication performance. This placement also would be optimal for scaling across geographies or data centers. This configuration also ensures high availability for a grid container server failure in a local zone (synchronous replica), and ensures high availability in a complete data center failure (asynchronous replica).

Typically, catalog servers are placed in each data center or zone, and the catalog servers synchronize their object/shard routing information. The catalog service must be clustered for high availability in every zone. The catalog service retains topology information of all of the containers and controls shard placement and routing for all clients.

This flexibility assures the availability of data to the application regardless of its zoned location. The catalog service provides up-to-date routing information about the location of an object should the object not be found in the zone with the closest proximity to the application container.

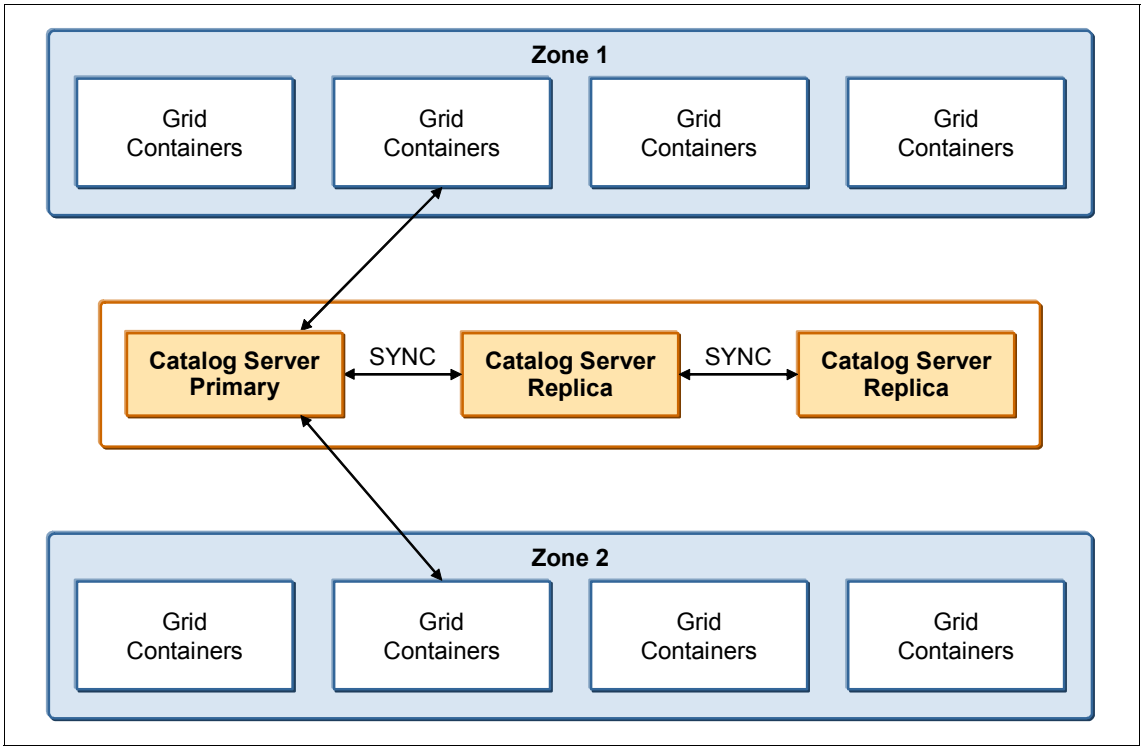


Figure 4-12 Zone placement

4.4.1 Zone-based routing

WebSphere eXtreme Scale provides a mechanism for clients to set preferences on how their requests are routed. eXtreme Scale supports a routing preference for zones, local host, and local process. This preference applies to both hash-based fixed partitions and per-container partitions.

Proximity-based routing provides the capacity to minimize client traffic across zone boundaries, to minimize client traffic across machines and to minimize client traffic across processes.

4.5 Configuration and management of the grid

When the design of the grid application and infrastructure has been determined, that information is shared among the grid clients and servers by means of configuration files. The files and their exact contents are specific to the application and the topology that is being supported.

4.5.1 Configuration of a local in-memory grid

In its most basic form, it is possible to create and use an eXtreme Scale grid to cache data locally in any Java process. In this instance, the grid is not shared with any other process and does not have the advanced features of replication, scalability, etc. It is still necessary to define the grid structure to eXtreme Scale at runtime. This can be done via a configuration XML file or via explicit API calls.

Since there are no replication settings and all of the data is cached locally, one must only define and assign a name to the grid and any BackingMaps that the grid will hold. Once it is initialized the client can store and reference data in the grid using the standard grid APIs.

4.5.2 Configuration of a distributed grid

A more typical case is a grid that is distributed among many servers. In this instance it is necessary to define more information about the environment in which the grid is running. This is done with two XML configuration files that are passed to the eXtreme Scale container during initialization.

The first configuration file is the same as that can be used in a local in-memory grid scenario. In it, the grid and its associated BackingMaps are defined by name.

The second configuration file defines the deployment topology and constraints for the grid. Due to the elastic nature of the grid environment, it is not necessary to define server names or port information as part of the deployment XML file. The deployment file is associated to the grid configuration file and specifies options such as how many partitions the application will use, the number of synchronous and asynchronous replicas that are required for each map set, and the number of initial containers that must be available to support the grid. The configuration of zones is also done as part of the deployment configuration.

The catalog service is responsible for placing the grid's primary and replica shards as the container servers become available.

4.5.3 Configuration of an eXtreme Scale client

The client to the grid may choose to define or modify configuration settings based on its own requirements. This can also be done programmatically as well as with XML files. Some of the options available to the client are to register additional call backs or plug-ins to the grid. Among other things, this allows a client to be informed of events in the grid's near cache (if one is configured), or to modify the evictor settings for the grid's near cache (if one is configured).

4.5.4 Management of grid in a non-WebSphere environment

WebSphere eXtreme scale supports most JSE and JEE runtimes. The eXtreme Scale application in a non-WebSphere environment must ensure that the appropriate configuration XML files are available as well as the runtime JAR files. WebSphere eXtreme Scale provides a script that can be used to start catalog or container servers for the grid. In each case the hostname and port of the catalog server (or servers) must be passed as an argument on the command line. The full set of catalog service endpoints must be specified when starting a catalog server. Container servers must also have the configuration XML files passed as command line arguments.

4.5.5 Management of the grid in a WebSphere Application Server environment

When WebSphere eXtreme Scale is integrated with a WebSphere Application Server installation, additional benefits are provided. By configuring custom properties, it is possible to have any WebSphere process host a catalog service grid.

The WebSphere runtime will make the eXtreme Scale JAR files available and will initialize an application server as a container server if the grid configuration and deployment XML files are present in the enterprise application. The grid container is automatically started when the enterprise application starts and is stopped when the enterprise application is stopped.

4.6 APIs used to access the grid

There are two APIs that can be used to access data in the grid:

- ObjectMap
- EntityManager

The following sections will provide a short description and the corresponding benefits and limitations of each API.

4.6.1 ObjectMap API

The ObjectMap API provides a transactional map-based API that allows typical CRUD (Create, Read, Update and Delete) operations to data in the grid. A com.ibm.websphere.objectgrid.ObjectMap contains the ObjectGrid session and the transaction data for the client application. That stored data is either targeted for or retrieved from the BackingMap. For more information about the ObjectMap programming API, see the following Web page:

http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r0/topic/com.ibm.websphere.extremescale.prog.doc/cxsobjectmapapi.html

Benefits

ObjectMaps provide a simple and intuitive approach for the application to store data. An ObjectMap is ideal for caching objects that have no relationships involved. ObjectMaps are like Java maps, allowing data to be stored as key/value pairs. Access is easy and fast with primary-key based access. Because ObjectMaps are like Java maps, they should be familiar to programmers who are familiar with java.util.Map interface.

Limitations

ObjectMaps are not ideal if object relationships are involved in your data storage scheme. There are also performance considerations, due to reliance on Java serialization.

Serialization versioning

WebSphere eXtreme Scale depends heavily on Java object serialization to transfer object instances between JVMs. Keep in mind that the objects in the grid might exist there for a long time, perhaps for months or years, but new releases of an application occur much more frequently.

A new release may not be able to read objects from the grid that have been placed there by a previous release. This can be caused by incompatible changes in the class (for example, changing the type of an attribute from “String” to

"Integer"). A simple solution would be to completely bring down the grid for re-deployment and bring it back (hopefully with pre-loading) afterwards.

But there is a better solution. Java offers a sophisticated class versioning mechanism for serialization. A good starting point is the Java serialization specification, especially Chapter 5 "Versioning of Serializable Objects," available at the following Web page:

http://java.sun.com/j2se/1.5.0/docs/guide/serialization/spec/serialTOC.html

Be sure to fully understand the concept of *compatible* and *incompatible* changes.

The API documentation is also worth reading and can be found at the following Web page:

http://java.sun.com/j2se/1.5.0/docs/api/java/io/Serializable.html

Leading practices for serialization versioning:

- Ensure every class that is used to store objects in the grid (that is, keys and values) has an explicitly declared `static final long serialVersionUID` with a value generated on initial release.
- Establish a build process that can verify that a new version of your application can still deserialize all object types serialized by the previous version. This can be accomplished by storing serialized sample objects in the version control system, and de-serializing these objects at build time to verify integrity. This could be implemented as a JUnit Test, for example.
- Make sure the object classes are available to the grid container.

4.6.2 EntityManager API

The EntityManager API is a simple and intuitive programming model for accessing data in a grid. As an alternative to the ObjectMap API, objects are represented as entities, which allows relationships and may optimize performance. Relationships are defined in a schema or through Java annotations. The EntityManager API uses the existing map-based infrastructure, but it converts entity objects to and from tuple objects before storing or reading them from the map.

Benefits

The EntityManager API is ideal when object relationships are involved. It provides an easy way to interact with a graph of related objects or object graphs. There is optimized performance for queries and for loading objects from the backing data source. The EntityManager API also may be easier to use due to its reliance on POJO-style programming, which has been adopted by most enterprise application architectures.

Limitations

The EntityManager requires the definition of schema in an entity descriptor XML file. Applications may have to be re-architected to avoid complex relationships between objects and to ensure that there is an absolute relationship between a root and its branches (this relationship is known as a constrained tree schema). Two applications may not be able to share a grid if both of the applications use different objects for the same data. Also, complex queries may not perform well due to the partitioned nature of the data. Although the EntityManager API is easier to use than the ObjectMap API, it is slower than ObjectMap in general due to the EntityManager converting entity POJO objects to and from tuple objects.

4.7 A simple example

Figure 4-13 on page 65 shows the different components that make up a grid and illustrate the flow from the application to the grid. This example is taken from a simple distributed cache topology. Figure 4-13 on page 65 shows an application that retrieves an object for a key from the grid. It updates the object's value and commits the change to the grid. Following Figure 4-13 on page 65 is an explanation of the process.

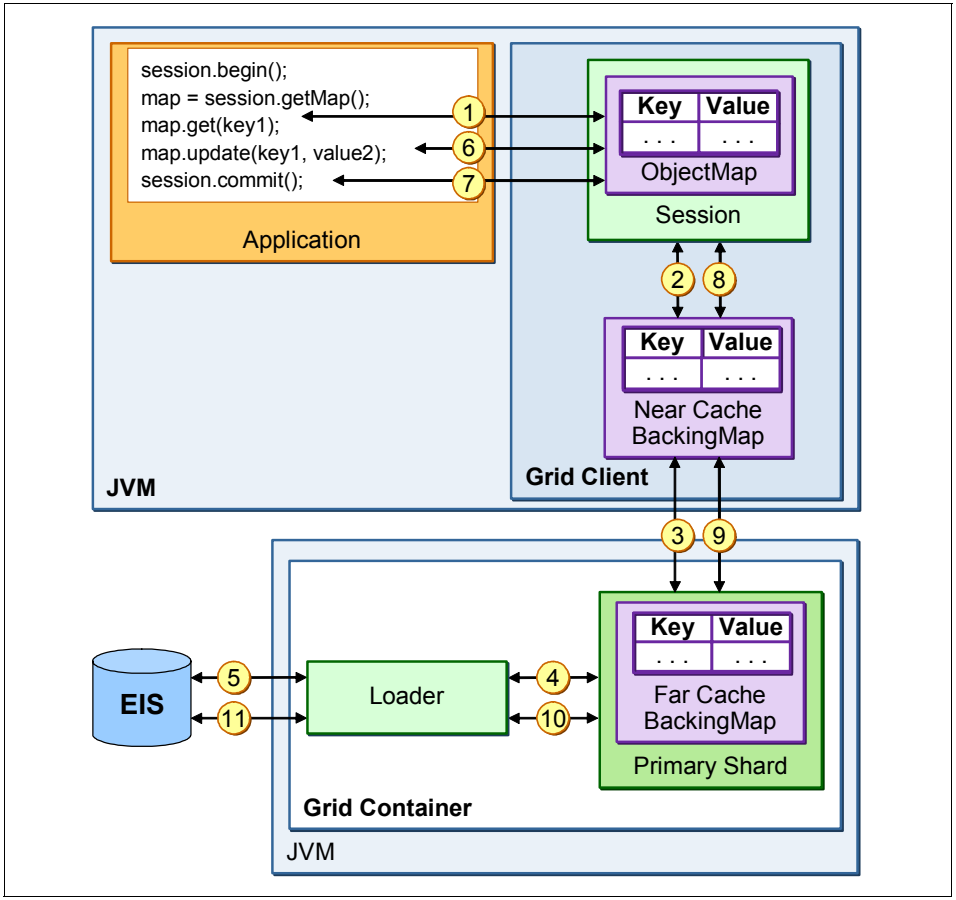


Figure 4-13 Component interactions for simple grid access

1. The application requests an object from the grid for a given key. As this request is the first time that the object has been accessed, it is not yet present in the local session ObjectMap.
2. The grid client uses the ObjectMap to attempt to retrieve the object from the client near-cache (which is actually a local instance of a backing map). Assuming no other thread in the application has accessed this particular object, there is also a miss here.
3. The grid client determines the partition number in which the object for the key should reside. Usually, the primary shard of that partition is accessed. If replicaReadEnabled was configured, a replica shard might be accessed instead, depending on the lock strategy used. If the grid container is

collocated with the application in the same JVM or host, this adjustment could increase speed. In either case, the grid client connects to the selected shard and asks for the object.

4. The shard checks whether an object for the requested key is present in its backing map. If so, the object will be returned to the caller. If not, and there is a loader for the map, the shard will call the loader to fetch the object. Otherwise, the shard will return null.
5. The loader is invoked to bring in the data from the backing data store, for example, using JDBC or JPA, and the object is constructed, stored in the backing map, and returned to the caller.
6. The client changes the object for a given key and puts it back in the ObjectMap using the update() method. Note that the changes occur only in the session's local copy of the object. Other sessions accessing that object cannot see this change yet.
7. The client commits the changes, which sets off a sequence of calls to propagate the changes into the grid and down to the backing data store.
8. The object is updated in the client near cache.
9. The update is propagated to the backing map of the primary shard.
10. The backing map calls the loader to update the value. By default, this updating is a synchronous operation. If write-behind is configured it occurs asynchronously after some delay.
11. The backing map updates the entry for the key. If replicas are configured, they are now contacted and informed of the update (not shown in this example). This can happen synchronously, meaning the operation will return only after the replica has been updated successfully) or asynchronously, in which case the operation will return immediately.
12. The Loader commits the changes to the back-end and returns to the caller.

4.8 WebSphere eXtreme Scale development environments

Developing WebSphere eXtreme Scale applications is not significantly different from developing any other Java applications. A good integrated development environment (IDE) such as IBM Rational® Application Developer or Eclipse is recommended. The application that is going to use eXtreme Scale could be a stand-alone Java application or one that requires an application server environment as an execution environment. WebSphere eXtreme Scale does not change those requirements. The required eXtreme Scale JAR files must be made available to the development project.

There are some special development provisions that make developing and testing eXtreme Scale applications easier. One example is that an application that is expected to have many container servers and partitions in its production runtime environment may be tested with far fewer containers by overriding the configuration for the number of partitions or containers. It is also possible to have eXtreme Scale place replicas on the same machine for testing purposes (which it will not do normally). In production the developmentMode in the deployment descriptor should be set to false. In a WebSphere Application Server development environment, it is possible to have the WebSphere deployment manager or node agents perform the role of catalog servers to minimize the number of processes that must be running to test the applications.

4.9 Scalability sizing considerations

WebSphere eXtreme Scale provides a scalable framework with a choice of runtime topologies to extend linearly the scalability of a data-driven transactional application. That is, the grid can grow elastically and transparently to grid operation with increase in demand and data without increased overhead. This section provides some high-level guidelines for sizing aimed at ensuring the scalability and reliability of a grid.

4.9.1 Heap size and the number of JVMs

This is probably the most common, and important consideration. While the eXtreme Scale grid offers linear scalability, insights into grid deployment, data size, and application patterns suggest the size and volume of JVMs that make up the grid. This insight is also an important data point to facilitate future growth of the grid. For instance, the grid deployment factors such as estimated partition size, number of partitions, number of synchronous and asynchronous replicas, desired availability, zoned deployment over geographies, and so forth, should be considered while deciding the total JVMs and size (in terms of memory allocation) of JVMs that make up the grid.

4.9.2 Number of grids

A single grid infrastructure can support many logical grid instances, each containing a distinct set of maps and data. A single application can connect to multiple grid instances if required. In general, each application should have its own dedicated grid instance, except in cases where data is shared between applications.

Put a different way, multiple applications and multiple grids can share the same catalog service. In general, logical sets of grids used by one or more applications should have its own dedicated catalog service to maintain a separation of applications so that one application's problems do not cause failures in the other application.

4.9.3 Catalog servers

The catalog service plays a central role in eXtreme Scale grid management. It is therefore vital to plan for catalog service high availability and sizing requirements. The decision on the number of catalog servers in the catalog service depends on overall grid size, desired high availability, and zones configuration. As a rule of thumb, there should be at least two catalog servers for high availability, and at least one per zone.

4.9.4 Sizing for growth

The appeal of WebSphere eXtreme Scale is the ability to scale linearly with growth. It is therefore vital to factor in the growth imperatives for ease of grid infrastructure administration and for accommodation of growth. Growth imperatives include decisions and considerations regarding grid topology, hardware requirements and availability, and managed or stand-alone grid environment. The driving factors are the resource (hardware and software) availability and the set of tasks involved in adding grid containers to expand the grid on demand. This imperative may also include the decision points and operational procedures required to add to the grid capacity.

It is particularly important to set the number of partitions high enough to allow for future growth. Once the application is deployed, the number of partitions cannot be changed without restarting the entire grid. This means that beyond the point where there is one only one shard on each grid container JVM, adding more JVMs will not provide any benefit. A good rule of thumb is 10 shards per grid container JVM.

4.10 Common topology configurations

Before designing your topology, it is important to consider what type of software you will install on your servers to house your grid. There are two types of servers, stand-alone and managed (by a deployment manager). You can use a combination of both type of servers.

4.10.1 Managed grid

You can use WebSphere Application Server Network Deployment servers to host your eXtreme Scale container JVMs. The main benefit of this configuration is that you can more easily manage your environment using the administrative capabilities available in the Network Deployment product. The clustering and high availability management features offered by the managed environment can be exploited. Grid extensibility becomes relatively easier in a managed environment, as creating grid servers to extend the grid is only a matter of a few clicks (as long as the capacity supports the grid expansion). Configuring zones in a Network Deployment environment is easily done using the node group capabilities. Additionally, the commonly available monitoring tools that may already be employed to monitor the performance and availability of your environment can be used to monitor the grid servers.

4.10.2 Stand-alone grid

Stand-alone servers can use the JSE implementation of your choice. The main benefit of this configuration is the use of a less expensive JSE environment for grid containers. Environments that already use JSE for their applications may also be inclined to use JSE stand-alone containers. In this case, the inclusion of WebSphere eXtreme Scale as a platform may be the only new addition. While there are obvious cost advantages to using stand-alone servers, the downside is the possible lack of availability management and monitoring solutions for the JSE containers that host the grid servers.

A common scenario is the use of a Network Deployment-managed environment to host enterprise applications and the use of a readily available JSE runtime environment as the grid layer.

Regardless of which type of server is chosen, the core capability of WebSphere eXtreme Scale, which is a transactional, secure, and scalable application cache fabric is available to be exploited. The issues around management and monitoring is environment-specific and a personal choice. Each topology discussed below can be implemented on either type of server or using a combination of servers of both types.

4.10.3 Local cache topology

In the local cache topology (shown in Figure 4-14), the application logic runs in the same JVM as the data in the grid. Each application will only access the local ObjectGrid instance to store or retrieve data from its cache. WebSphere eXtreme Scale, in this case, is used as a simple local cache.

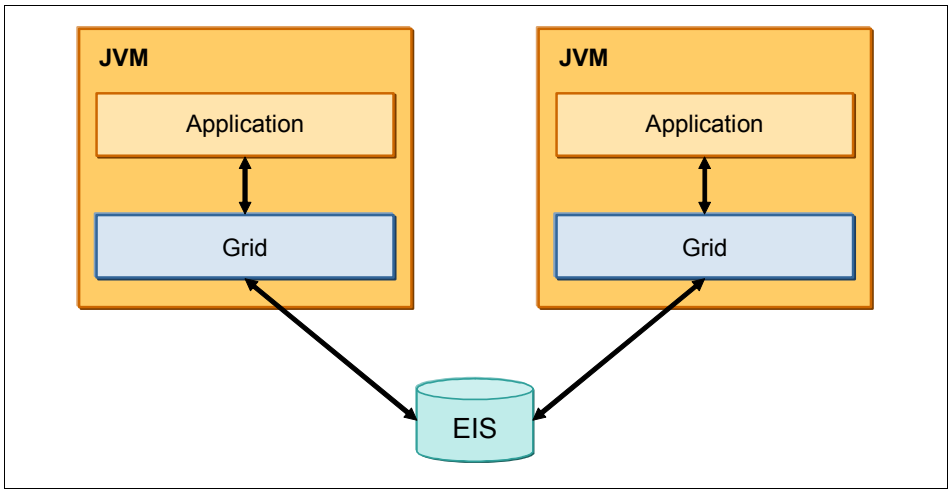


Figure 4-14 Local cache topology

This topology can perform faster than using a database if the needed data can be found in the local ObjectGrid. This avoids a remote procedure call (RPC) to the back-end data store. Using WebSphere eXtreme Scale as a local cache can also reduce the load on your back-end data store. This topology is not recommended for fault tolerance or high availability. A big problem with this approach as shown in the figure is if you have more than one cache for the data. You then have to keep the caches in sync. Suppose an object is cached in both JVMs, and is changed in one of them. The other cache is now stale, because it has an old value of the object. How does the other cache get an updated version of this object? This is really not a problem you want to deal with. It also scales poorly as you add applications with local caches. Use some of the following topologies instead to have a shared cache and avoid the problem altogether. See 4.11, “Handling of stale caches” on page 74 for additional stale cache issues. This topology works really well if the applications are only reading the data, not making any changes.

4.10.4 Collocated application and cache topology

In the collocated application and cache topology shown in Figure 4-15, the application logic runs in the same JVM as the data in the grid. However, the data stored in the grid is spread across all the JVMs that have WebSphere eXtreme Scale installed and configured.

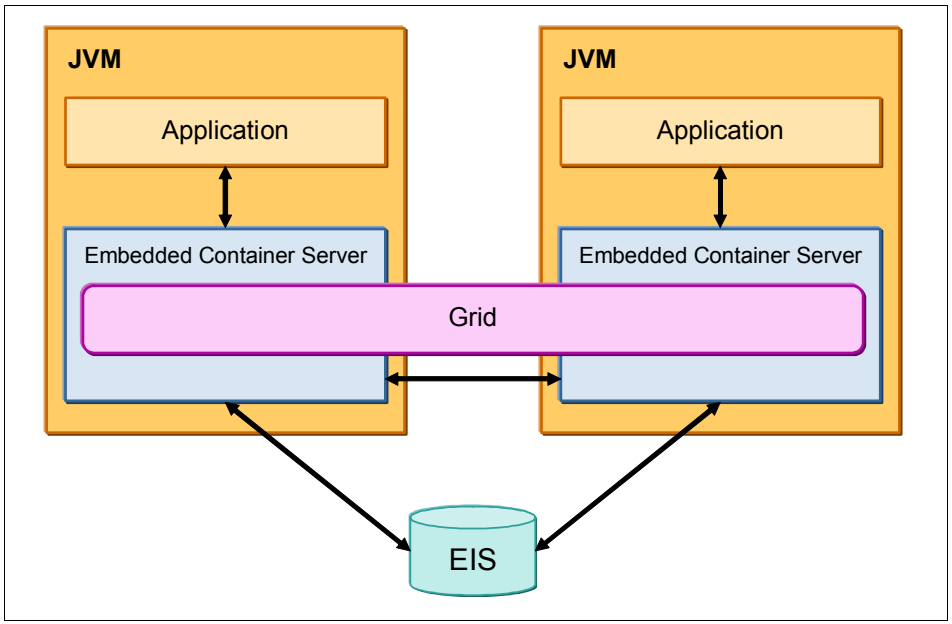


Figure 4-15 Co-located application and cache topology

This topology can be faster than using a database because an application can take advantage of the grid near-cache to compensate for the RPC calls made when the requested data is stored on another server in the grid, or can only be found in the back-end datastore. This topology can also reduce the load on your back-end datastore.

With this topology, replica shards that sit in a JVM other than the primary can be used to ensure fault tolerance and high availability.

4.10.5 Distributed cache topology

In the distributed cache topology shown in Figure 4-16 on page 72, the application logic runs on application servers separate from the grid servers. The application servers host a grid client that communicates with the grid servers to access data from the far cache. The data stored in the grid is spread across all

the JVMs that have WebSphere eXtreme Scale installed and configured. In this case, the WebSphere eXtreme Scale distributed grid is more like a data service proxy for the back-end system. This topology can also reduce the load on your backing data store.

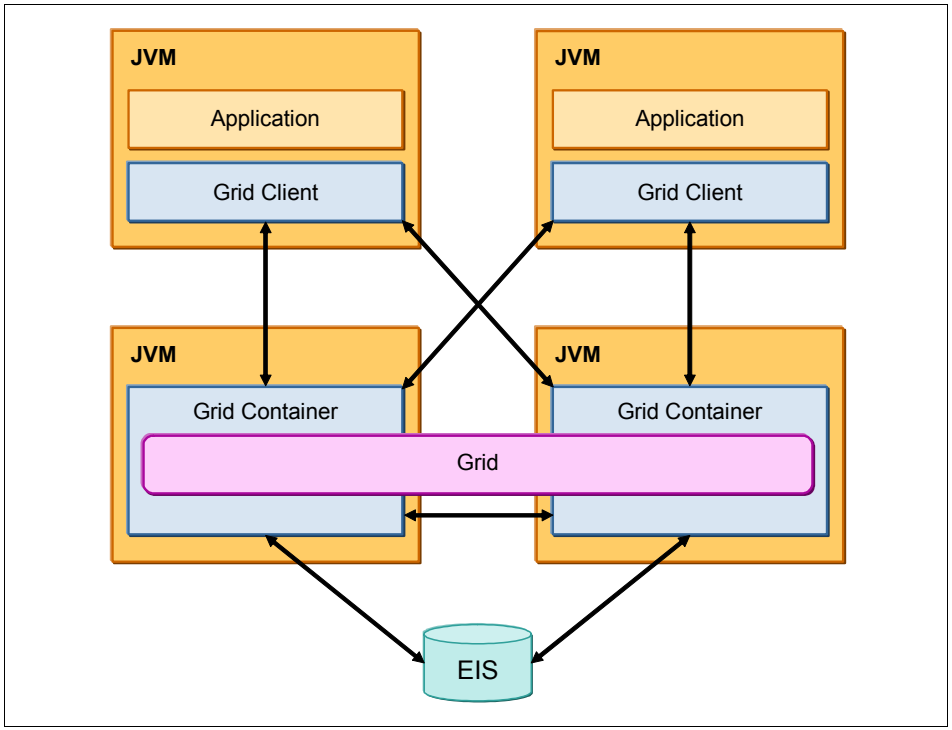


Figure 4-16 Distributed cache topology

There are several benefits to using this topology. Replica shards can sit in JVMs separate from the primary shards to ensure fault tolerance and high availability. The application servers can be restarted without interrupting the availability of the grid servers. If you want to employ clustering, you should cluster the application servers separately from the grid servers, so the application cluster can be restarted without affecting the grid layer.

4.10.6 Zone-based topology

There are cases when enterprise computing environments would like to distribute their data cache across geographies for high availability and disaster recovery motives. Zone support provides much needed control of shard placement in the WebSphere eXtreme Scale-enabled grid. Zones, by definition, can be considered as a set of grid containers that belong to a domain or exist

within some boundary. Multiple zones can be envisioned to exist across WANs and LANs, or even in the same LAN, but one zone is not intended to span across a WAN. Instead, multiple zones should be defined across a WAN, combining to form one single grid. Such a topology includes the following advantages:

- High availability of data cache across geographies
- Proximity of data to the application
- Controlled rule-based replication
- Primary and replica can be placed in different zones, satisfying the disaster recovery requirements.

The replication of data in real time, such as HTTP session data and application data, was a concern in the past due to cost of network and computing resources. The effort and costs involved in achieving this type of replication outweighed the potential benefits. Slower network connections mean lower bandwidth and higher latency connections. Zone-based replication factors in the possibility of network partitions, latencies, network congestion, and other factors. A WebSphere eXtreme Scale grid adjusts to this unpredictable environment in the following ways:

- Limiting heartbeat to reduce traffic and processing
- Exploiting the catalog service as a centralized location service

Figure 4-17 illustrates a zone-based topology.

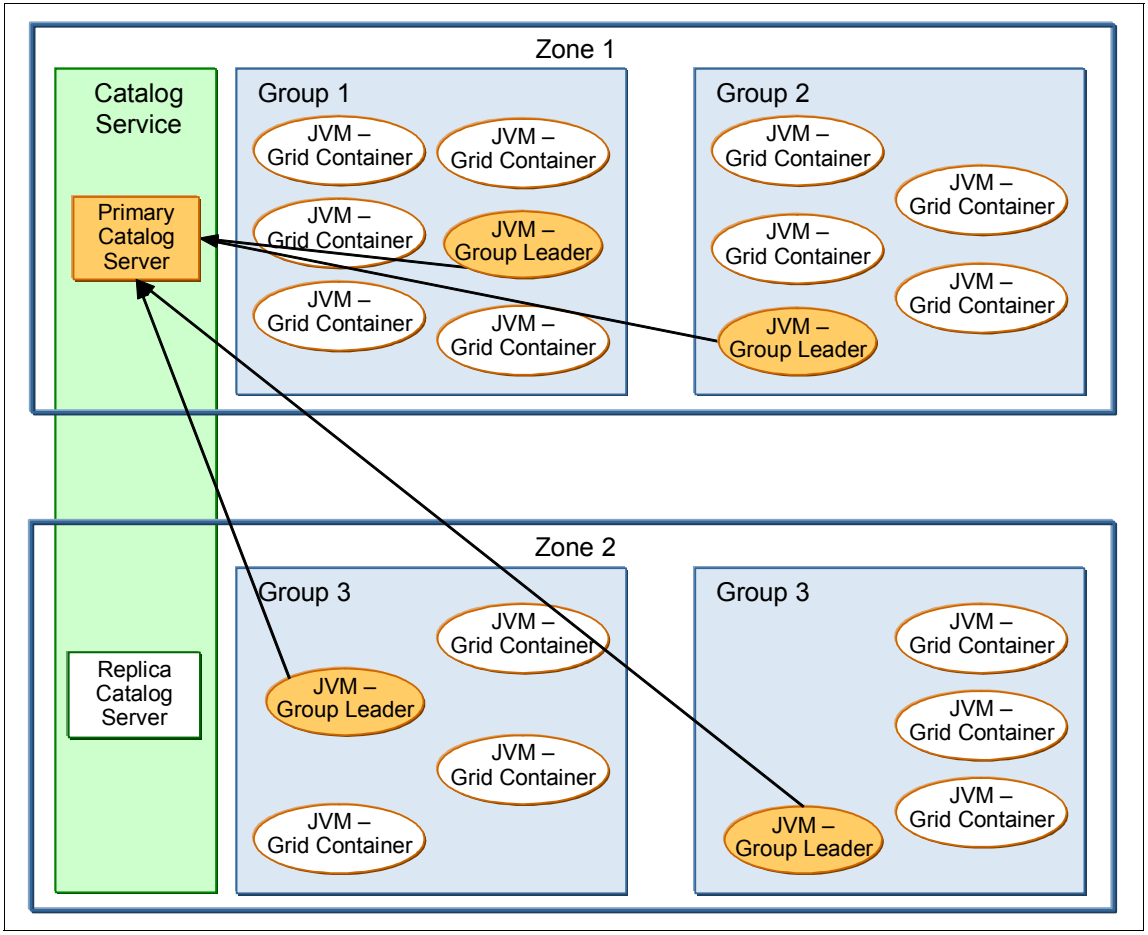


Figure 4-17 Zone-based topology

4.11 Handling of stale caches

Whenever an application uses cache or grid functionality in conjunction with a hardened backing data store, the problem of stale data in the cache must be considered. For example, an application reads old data from the cache, but the data has already been updated in the backing data store. This situation is called a dirty read because of the stale cache data. This situation can have a severe impact for the application and the business it supports. It occurs frequently when many external changes (that is, not managed by the eXtreme Scale applications) are applied to the backing data store.

The requirements on cached data accuracy have to be determined by finding answers to the following questions:

- How acceptable is a dirty read?
- What are the consequences if one occurs?
- What is the maximum acceptable time between an update in the back-end store and an update in the cache?

The answer to these questions depend on the type of data and the type of use cases the application drives against it. The answers can range from “dirty reads can occur and do not really matter” to “dirty reads are not acceptable in any case.”

As an example, note the shadow data base in the figures in the first chapter. Many organizations use shadow data bases for ad-hoc queries. The shadow data base is stale. You live with the staleness. It is typically only a few minutes stale, and you live with this delay not to slow down the operational applications that are updating the data.

Accuracy requirements have to be established for each entity that is stored in the cache. Different entities might have different requirements for a single application. There are several different approaches available with WebSphere eXtreme Scales to deal with this issue.

4.11.1 Simply tolerate

If requirements permit, dirty reads can simply be accepted as a matter of fact.

4.11.2 Use time-based eviction strategies

Use a time-based eviction strategy (for example, after 24 hours or every day at 04:00) to establish an upper bound for the dirty read time.

This strategy can cause peak loads at the backing data store when a lot of invalidated data is re-read at a certain point in time. It can cause performance degradation because of a lot of cache misses after a cache invalidation occurred.

4.11.3 Cache polls the database for updates in regular intervals

The cache application polls the backing data store for the latest changes. The application can either invalidate the objects from the cache, or directly update it. Use information provided by the backing data store to determine changes. Examples can be a time stamp column “last modified” that denotes the point in time. In this case, it is easy to select all rows that have been changed since the

last time the grid was updated from the backing data store. It is quite common for a business application to have a “last modified” time stamp for audit reasons that it can use.

eXtreme Scale supports polling the database for changes for JPA using the TimeBasedDBUpdater interface. When JPA is not used, a simple WorkerThread can be spawned by the application to poll changes from the database and update the grid.

This approach does not invalidate objects that have been deleted from the backing data store. Deleted objects are hard to select from a database for obvious reasons.

Stale data can still exist with polling-based invalidation. There can be quite a large time period between the time data changes and the next poll, during which clients will get stale data.

4.11.4 Use JMS publish/subscribe to propagate changes

Stale data can arise when several clients use a near cache and have local copies for the same key, and one client modifies this data. By default, the other clients are not notified of the modification, thus they have a stale object in their local cache.

JMS messaging, especially when using publish/subscribe topics, can be used to push changes from the grid to near caches at the client. eXtreme Scale supports this feature using the JMSObjectGridEventListener class. See the information about the JMSObjectGridEventListener class at the following Web page:

http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r0/topic/com.ibm.websphere.extremescale.javadoc.doc/topics/com/ibm/websphere/objectgrid/plugins/builtins/JMSObjectGridEventListener.html

Leading practice: Use small near caches with quick eviction strategy.

Stale near caches can be avoided by configuring a small cache size and an eviction strategy that promptly removes the objects from the near cache. Keep in mind that there is a large grid behind the near cache, and it is not expensive to get the data from the grid again. If the data in the backing store is not changed frequently, near caches can be effective. When the data in the backing store is changing frequently the benefits from a near cache can be limited.

4.11.5 Make sure no external changes to the backing store occur

This approach prohibits external changes to be directly applied to the backing store. Instead, every change to the data is only allowed through applications that interact with the grid. This approach can be difficult if other components or applications that are not aware of the grid have write access to the backing store. For example, subsystems, interfaces to external systems, data cleansing scripts, and so forth would not be allowed to change data.

4.11.6 Make sure all external change processes notify the grid

Instead of prohibiting all external changes as in the previous approach, this approach envisions that the grid is notified of all external changes. All database scripts or other external modifications need to be equipped with a grid client that notifies the grid about changes. The easiest notification would be to invalidate the data in the cache. A more sophisticated solution might be able to directly update the data. This allows for queries in the grid since the grid is maintaining all entries that are stored in the database.

The downside of this approach is that it might be difficult to ensure that all scripts, applications, and so forth, are aware of the grid.

4.11.7 Push the changes from the back-end store up to the grid

For SQL databases, post-insert, update, and delete triggers can be used to invalidate or update the grid. This approach requires a grid client to be embedded into the database, which is not a problem as long as the database supports triggers that are implemented in Java.

The triggers are usually executed before the transaction is committed. This circumstance can lead to an unnecessary cache invalidation when the transaction is rolled back in the end. But this invalidation is acceptable when dirty reads cannot be tolerated. It is better to have an unnecessary invalidation than a stale object.

4.11.8 Reload the grid in off hours

The backing store updates can be performed off hours and the grid can be cleared and pre-loaded again with the updated data. The downside of this approach is that with a large data set, the preload time is rather large and if your application does not have off hour down time it is not possible.

4.12 WebSphere real time support

WebSphere Real Time can be used with WebSphere eXtreme Scale. When WebSphere Real Time is enabled the application will have stable and consistent response times and throughput. This is accomplished with the WebSphere Real Time garbage collection cycles being more predictable.

WebSphere eXtreme Scale applications create many objects with each transaction. These objects are related to application as well as internal components such as logging and sessions. Without WebSphere Real Time, the garbage collection interval is unbounded and can in some instances cause transaction response times to take an excessive amount of time (hundreds of milliseconds or one or more seconds). If consistent response times are required you may wish to consider using WebSphere Real Time along with WebSphere eXtreme Scale.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 80. Note that some of the documents referenced here may be available in softcopy only.

- *User's Guide to WebSphere eXtreme Scale*, SG24-7683

Online resources

These Web sites are also relevant as further information sources:

- WebSphere eXtreme Scale product home page

  http://www.ibm.com/software/webservers/appserv/extremescale

- IBM Education Assistant

  http://www-01.ibm.com/software/info/education/assistant

- WebSphere eXtreme Scale Information Center

  http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r0/index.jsp

- WebSphere Education

  http://www.ibm.com/websphere/education

- Course description: Developing Applications for WebSphere eXtreme Scale V7

  http://www-304.ibm.com/jct03001c/services/learning/ites.wss/us/en?pageType=course_description&courseCode=WA972

© Copyright IBM Corp. 2009. All rights reserved.

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM WebSphere eXtreme Scale V7: Solutions Architecture

Product features

Application scenarios

Architectural overview

The IBM WebSphere eXtreme Scale product provides a powerful, elastic, high-performance, and scalable in-memory data grid. This paper will help IT architects understand how this data grid can be used to enhance application performance and scalability. It introduces the concepts behind eXtreme Scale and shows how it addresses the challenges of scalability and throughput found in today's business applications.

You will find information about entry points for integrating WebSphere eXtreme Scale into your environment, and a decision tree to help you select the features of eXtreme Scale that are especially suitable for improving performance in your environment.

This paper takes you through a number of application scenarios to illustrate the benefits that eXtreme Scale can provide. And finally, it provides an in-depth architectural discussion to help you understand how the product works and how it is integrated into an existing application environment.

This paper is a follow-on to *User's Guide to WebSphere eXtreme Scale*, SG24-7683, updating the architectural content for WebSphere eXtreme Scale V7. The technical portion of that book is still relevant and can be used as a guide to the implementation of eXtreme Scale.

REDP-4602-00

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks